UVM PROJECT

SPI-slave with a single-port RAM

Submitted by:

Mohammed Essam
Abdulrahman Dakrory
Youssef Seyam

Contents

Part 1: UVM Environment for SPI-Slave

1. Code

1.1 Top

```
import uvm_pkg::*;
`include "uvm_macros.svh"
import SPI_slave_test_pkg::*;
import SPI_slave_shared_pkg::*;

module top();
   bit [2:0] cs;
   bit [3:0] counter;
  bit clk;
  // Clock generation
  initial begin
    forever begin
      #5 clk=~clk;
    end
  end
  // Instantiate the interface and DUT
   SPI_slave_if SPI_slaveif(clk);

   SPI_Slave_Golden Golden(SPI_slaveif);

    SLAVE DUT (SPI_slaveif);
    assign cs=DUT.cs;
    assign counter=DUT.counter;

  // run test using run_test task
 initial begin
  uvm_config_db #(virtual
SPI_slave_if)::set(null,"uvm_test_top","SPI_slave_if",SPI_slaveif);
  run_test("SPI_slave_test");
 end

endmodule
```

1.2 SPI_slave_test

```
package SPI_slave_test_pkg;
  import uvm_pkg::*;
  `include "uvm_macros.svh"

  import SPI_slave_env_pkg::*;
```

```
  import SPI_slave_agent_pkg::*;
  import SPI_slave_config_pkg::*;
  import SPI_slave_main_seq_pkg::*;
  import SPI_slave_rst_seq_pkg::*;
  import SPI_slave_shared_pkg::*;
  class SPI_slave_test extends uvm_test;
    `uvm_component_utils(SPI_slave_test)

    SPI_slave_env env;
    SPI_slave_config cfg;
    SPI_slave_main_seq main_seq;
    SPI_slave_rst_seq rst_seq;

    function new(string name = "SPI_slave_test", uvm_component parent =
null);
      super.new(name, parent);
    endfunction

    function void build_phase(uvm_phase phase);
      super.build_phase(phase);
      env = SPI_slave_env::type_id::create("env", this);
      cfg = SPI_slave_config::type_id::create("cfg");

    if(!uvm_config_db #(virtual
SPI_slave_if)::get(this,"","SPI_slave_if",cfg.SPI_slavevif)) begin
     `uvm_fatal("build_phase","test - unable to get the virtual
interface")
    end
      cfg.is_active = UVM_ACTIVE;
      uvm_config_db#(SPI_slave_config)::set(this, "*", "CFG", cfg);
      rst_seq  = SPI_slave_rst_seq::type_id::create("rst_seq");
      main_seq = SPI_slave_main_seq::type_id::create("main_seq");
    endfunction

    task run_phase(uvm_phase phase);
      phase.raise_objection(this);
      `uvm_info("RUN_PHASE", "Starting reset sequence", UVM_LOW)
      rst_seq.start(env.agt.sqr);

      `uvm_info("RUN_PHASE", "Starting main sequence", UVM_LOW)
       main_seq.start(env.agt.sqr);

      `uvm_info("RUN_PHASE", "Test finished", UVM_LOW)
      `uvm_info("RUN_PHASE", $sformatf("Total errors: %0d", error_count),
UVM_LOW)
```

```
      `uvm_info("RUN_PHASE", $sformatf("Total correct: %0d",
correct_count), UVM_LOW)
      phase.drop_objection(this);
    endtask
  endclass : SPI_slave_test

endpackage : SPI_slave_test_pkg
```

1.3 SPI_slave_env

```
package SPI_slave_env_pkg;
  import uvm_pkg::*;
  `include "uvm_macros.svh"
  import SPI_slave_agent_pkg::*;
  import SPI_slave_scoreboard_pkg::*;
  import SPI_slave_coverage_pkg::*;

  class SPI_slave_env extends uvm_env;
    `uvm_component_utils(SPI_slave_env)
    SPI_slave_agent agt;
    SPI_slave_scoreboard sb;
    SPI_slave_coverage cov;

    function new(string name = "SPI_slave_env", uvm_component parent =
null);
      super.new(name, parent);
    endfunction

    function void build_phase(uvm_phase phase);
      super.build_phase(phase);
      agt = SPI_slave_agent::type_id::create("agt", this);
      sb  = SPI_slave_scoreboard::type_id::create("sb", this);
      cov = SPI_slave_coverage::type_id::create("cov", this);
    endfunction

    function void connect_phase(uvm_phase phase);
      super.connect_phase(phase);
      agt.agt_ap.connect(sb.sb_export);
      agt.agt_ap.connect(cov.cov_export);
    endfunction
  endclass : SPI_slave_env

endpackage : SPI_slave_env_pkg
```

1.4 SPI_slave_agent

```
package SPI_slave_agent_pkg;
  import uvm_pkg::*;
  `include "uvm_macros.svh"
  import SPI_slave_config_pkg::*;
  import SPI_slave_driver_pkg::*;
  import SPI_slave_monitor_pkg::*;
  import SPI_slave_sequencer_pkg::*;
  import SPI_slave_seq_item_pkg::*;

  class SPI_slave_agent extends uvm_agent;
    `uvm_component_utils(SPI_slave_agent)

    SPI_slave_config cfg;
    SPI_slave_driver drv;
    SPI_slave_sequencer sqr;
    SPI_slave_monitor mon;
    uvm_analysis_port#(SPI_slave_seq_item) agt_ap;

    function new(string name = "SPI_slave_agent", uvm_component parent =
null);
      super.new(name, parent);
    endfunction

    function void build_phase(uvm_phase phase);
      super.build_phase(phase);
      if (!uvm_config_db#(SPI_slave_config)::get(this, "", "CFG", cfg))
        `uvm_fatal("CFG_NOT_FOUND", "You must set the CFG before building
the agent")

      if (cfg.is_active == UVM_ACTIVE) begin
        drv = SPI_slave_driver::type_id::create("drv", this);
        sqr = SPI_slave_sequencer::type_id::create("sqr", this);
      end
      mon = SPI_slave_monitor::type_id::create("mon", this);
      agt_ap = new("agt_ap", this);
    endfunction

    function void connect_phase(uvm_phase phase);
      super.connect_phase(phase);
      if (cfg.is_active == UVM_ACTIVE) begin
       drv.seq_item_port.connect(sqr.seq_item_export);
        // TODO: set driver's virtual interface
         drv.vif=cfg.SPI_slavevif;
      end
```

```
      // TODO: connect monitor vif
      mon.vif=cfg.SPI_slavevif;
      mon.mon_ap.connect(agt_ap);
    endfunction
  endclass : SPI_slave_agent

endpackage : SPI_slave_agent_pkg
```

1.5 SPI_slave_driver

```
package SPI_slave_driver_pkg;
  import uvm_pkg::*;
  `include "uvm_macros.svh"
  import SPI_slave_seq_item_pkg::*;
  import SPI_slave_config_pkg::*;

  class SPI_slave_driver extends uvm_driver#(SPI_slave_seq_item);
    `uvm_component_utils(SPI_slave_driver)
    virtual SPI_slave_if vif;
     SPI_slave_seq_item tx;
    function new(string name = "SPI_slave_driver", uvm_component parent =
null);
      super.new(name, parent);
    endfunction

    task run_phase(uvm_phase phase);

      forever begin
         tx = SPI_slave_seq_item::type_id::create("tx");
        seq_item_port.get_next_item(tx);
        // TODO: Drive DUT signals using tx fields
        vif.rst_n    = tx.rst_n;
        vif.MOSI     = tx.MOSI;
        vif.SS_n     = tx.SS_n;
        vif.tx_valid = tx.tx_valid;
        vif.tx_data  = tx.tx_data;
        @(negedge vif.clk);
        seq_item_port.item_done();
      end
    endtask
  endclass : SPI_slave_driver

endpackage : SPI_slave_driver_pkg
```

1.6 SPI_slave_monitor

```
package SPI_slave_monitor_pkg;
  import uvm_pkg::*;
  `include "uvm_macros.svh"
  import SPI_slave_seq_item_pkg::*;

  class SPI_slave_monitor extends uvm_monitor;
    `uvm_component_utils(SPI_slave_monitor)

    virtual SPI_slave_if vif;
    SPI_slave_seq_item tx;
    uvm_analysis_port#(SPI_slave_seq_item) mon_ap;

    function new(string name = "SPI_slave_monitor", uvm_component parent =
null);
      super.new(name, parent);
    endfunction
    function void build_phase(uvm_phase phase);
     super.build_phase(phase);
        mon_ap=new("mon_ap",this);
    endfunction
    task run_phase(uvm_phase phase);

     forever begin
       tx = SPI_slave_seq_item::type_id::create("tx");
       @(negedge vif.clk);
       // TODO: sample DUT outputs and fill tx fields
        tx.rst_n    = vif.rst_n   ;
        tx.MOSI     = vif.MOSI    ;
        tx.SS_n     = vif.SS_n    ;
        tx.tx_valid = vif.tx_valid;
        tx.tx_data  = vif.tx_data ;
        tx.rx_valid = vif.rx_valid;
        tx.rx_data  = vif.rx_data ;
        tx.MISO    = vif.MISO   ;

        tx.rx_valid_ref = vif.rx_valid_ref ;
        tx.rx_data_ref  = vif.rx_data_ref ;
        tx.MISO_ref    = vif.MISO_ref   ;

       mon_ap.write(tx);
       //`uvm_info("SEQ_ITEM", tx.convert2string(), UVM_MEDIUM)
      end
    endtask
  endclass : SPI_slave_monitor
```

```
endpackage : SPI_slave_monitor_pkg
```

1.7 SPI_slave_sequencer

```
package SPI_slave_sequencer_pkg;
  import uvm_pkg::*;
  `include "uvm_macros.svh"
  import SPI_slave_seq_item_pkg::*;

  class SPI_slave_sequencer extends uvm_sequencer#(SPI_slave_seq_item);
    `uvm_component_utils(SPI_slave_sequencer)
    function new(string name = "SPI_slave_sequencer", uvm_component parent
= null);
      super.new(name, parent);
    endfunction
  endclass

endpackage : SPI_slave_sequencer_pkg
```

1.8 SPI_slave_scoreboard

```
package SPI_slave_scoreboard_pkg;
  import uvm_pkg::*;
  `include "uvm_macros.svh"
  import SPI_slave_seq_item_pkg::*;
  import SPI_slave_shared_pkg::*;

  class SPI_slave_scoreboard extends uvm_scoreboard;
    `uvm_component_utils(SPI_slave_scoreboard)
    uvm_analysis_port#(SPI_slave_seq_item) sb_export;
    uvm_tlm_analysis_fifo#(SPI_slave_seq_item) sb_fifo;
    SPI_slave_seq_item rx;

    function new(string name = "SPI_slave_scoreboard", uvm_component
parent = null);
      super.new(name, parent);
    endfunction

    function void build_phase(uvm_phase phase);
      super.build_phase(phase);
      sb_export = new("sb_export", this);
      sb_fifo   = new("sb_fifo", this);
    endfunction
```

```
    function void connect_phase(uvm_phase phase);
      super.connect_phase(phase);
      sb_export.connect(sb_fifo.analysis_export);
    endfunction
    task run_phase(uvm_phase phase);
      forever begin
        sb_fifo.get(rx);
        golden_model(rx);
      end
    endtask

    // TODO: Implement golden_model tailored to your DUT's expected
behavior
    task automatic golden_model(input SPI_slave_seq_item tx);
       // Example: Log the received transaction
       if(tx.rx_valid_ref!=tx.rx_valid || tx.MISO_ref!=tx.MISO) begin
          error_count++;
          `uvm_error("SCOREBOARD", $sformatf("Mismatch in rx_valid:
expected %0b, got %0b", tx.rx_valid_ref, tx.rx_valid))
       end else begin
          `uvm_info("SCOREBOARD", $sformatf("rx_valid matches: %0b",
tx.rx_valid), UVM_LOW)
             if(tx.rx_valid==1) begin
                if(tx.rx_data_ref!=tx.rx_data) begin
                   error_count++;
                   `uvm_error("SCOREBOARD", $sformatf("Mismatch in
rx_data: expected %0h, got %0h", tx.rx_data_ref, tx.rx_data))
                end else begin
                   correct_count++;
                   `uvm_info("SCOREBOARD", $sformatf("rx_data matches:
%0h", tx.rx_data), UVM_LOW)
             end
        end
      end
     // `uvm_info("SCOREBOARD", $sformatf("Golden model called for: %s",
tx.convert2string()), UVM_LOW)
    endtask
  endclass : SPI_slave_scoreboard

endpackage : SPI_slave_scoreboard_pkg
```

1.9 SPI_slave_coverage

```
package SPI_slave_coverage_pkg;
  import uvm_pkg::*;
```

```
  `include "uvm_macros.svh"
  import SPI_slave_seq_item_pkg::*;

  class SPI_slave_coverage extends uvm_component;
    `uvm_component_utils(SPI_slave_coverage)
    uvm_analysis_port#(SPI_slave_seq_item) cov_export;
    uvm_tlm_analysis_fifo#(SPI_slave_seq_item) cov_fifo;
    SPI_slave_seq_item cov_item;

    covergroup cg;
     rx_data_cp: coverpoint cov_item.rx_data[9:8];
     SS_n_cp: coverpoint cov_item.SS_n {bins Write_bin_trans= (1=> 0
[*13]);
                                        bins Read_bin_trans= (1=> 0
[*23]);}
    MOSI_cp: coverpoint cov_item.temp_MO_data[10:8]
    {bins write_addr  = {3'b000};
    bins write_data  = {3'b001};
    bins read_addr   = {3'b110};
    bins read_data   = {3'b111};
     illegal_bins invalid_cmds = default;
    }
    SSN_cp : coverpoint cov_item.SS_n {
    bins high = {1'b1};
    bins low  = {1'b0};
  }
  // Cross coverage between SS_n and MOSI patterns
  SSN_MOSI_cross : cross SSN_cp, MOSI_cp {

    // Legal cases: SS_n high before valid MOSI commands
    bins write_addr_valid = binsof(SSN_cp.low) &&
binsof(MOSI_cp.write_addr);
    bins write_data_valid = binsof(SSN_cp.low) &&
binsof(MOSI_cp.write_data);
    bins read_addr_valid  = binsof(SSN_cp.low) &&
binsof(MOSI_cp.read_addr);
    bins read_data_valid  = binsof(SSN_cp.low) &&
binsof(MOSI_cp.read_data);

    // Exclude illegal or irrelevant cases (e.g., SS_n=1 with commands)
    ignore_bins irrelevant = binsof(SSN_cp.high) && binsof(MOSI_cp);
  }
    endgroup : cg
```

```
    function new(string name = "SPI_slave_coverage", uvm_component parent
= null);
      super.new(name, parent);
      cg = new();
    endfunction

    function void build_phase(uvm_phase phase);
      super.build_phase(phase);
      cov_export = new("cov_export", this);
      cov_fifo   = new("cov_fifo", this);
      `uvm_info("coverage","coverage build_phase", UVM_MEDIUM)
    endfunction
      function void connect_phase(uvm_phase phase);
      super.connect_phase(phase);
      cov_export.connect(cov_fifo.analysis_export);
    endfunction

    task run_phase(uvm_phase phase);
    super.run_phase( phase);
      forever begin
        // cov_item= SPI_slave_seq_item::type_id::create("cov_item");
        cov_fifo.get(cov_item);
         cg.sample();
        // `uvm_info("coverage", cov_item.convert2string(), UVM_MEDIUM)
      end
    endtask
  endclass : SPI_slave_coverage

endpackage : SPI_slave_coverage_pkg
```

1.10 SPI_slave_config

```
package SPI_slave_config_pkg;
  import uvm_pkg::*;
  `include "uvm_macros.svh"

  class SPI_slave_config extends uvm_object;
    `uvm_object_utils(SPI_slave_config)

    // TODO: Add your virtual interface handle(s) here
    // Example:
     virtual SPI_slave_if SPI_slavevif;
    // TODO: Add other config fields (e.g. is_active, clock_period,
time_scale...)
    uvm_active_passive_enum is_active;
```

```
    function new (string name = "SPI_slave_config");
      super.new(name);
    endfunction
  endclass

endpackage : SPI_slave_config_pkg
```

1.11 SPI_slave_seq_item

```
package SPI_slave_seq_item_pkg;
  import uvm_pkg::*;
  `include "uvm_macros.svh"
  import SPI_slave_shared_pkg::*;

  class SPI_slave_seq_item extends uvm_sequence_item;
    `uvm_object_utils(SPI_slave_seq_item)

    // TODO: add randomized stimulus fields that match your DUT
  rand logic    rst_n;
  rand logic  MOSI , SS_n, tx_valid;
  rand logic  [7:0] tx_data;

  rand bit [10:0] MO_data; // 11-bit array for MOSI data
  static bit [10:0] temp_MO_data;
  static bit [1:0] cmd_type; // Example command type (write/read)
    // Observed fields (from monitor)
   logic     [9:0] rx_data;
   logic           rx_valid, MISO;
   bit [9:0] rx_data_ref;
  bit      rx_valid_ref;
   bit      MISO_ref;

    function new(string name = "SPI_slave_seq_item");
      super.new(name);
    endfunction

    function string convert2string();
    return $sformatf(
    "%s SS_n=%0b | MOSI=%0b | tx_valid=%0b | tx_data=0x%0h | MO_data=0x%0h
| cmd_type=%0b | rx_data=0x%0h | rx_valid=%0b | MISO=%0b",
    super.convert2string(),
    SS_n,
    MOSI,
    tx_valid,
```

```
    tx_data,
    temp_MO_data,
    cmd_type,
    rx_data,
    rx_valid,
    MISO
  );
endfunction


    // TODO: Add constraints tailored to DUT behavior
    constraint rst_n_mostly_deasserted{ rst_n dist{1:=97,0:=3};}
    constraint c3 {
      MO_data[10:8] inside {3'b000, 3'b001, 3'b110, 3'b111};
      MO_data[7:0]   inside {8'h00,8'hFF,8'hA5};
    }
    static bit [10:0] prev_MO_data;
     constraint my_c{
      MOSI     inside {0};
      SS_n     inside {1};
      tx_valid inside {0};
      (prev_MO_data[8]==0)-> MO_data[8]==1;
      (prev_MO_data[8]==1)-> MO_data[8]==0;
     }

   function void post_randomize();

      static int cycle_count = 0;
      prev_MO_data = temp_MO_data;
        tx_data = temp_MO_data [7:0];
      if (!rst_n) begin
        cycle_count = 0;
        SS_n        = 1;
        tx_valid    = 0;
        MOSI        = 0;
        return;
      end

          $display("temp_MO_data=%0h cycle_count=%0d",temp_MO_data
,cycle_count);
        if (cycle_count == 0)begin
          SS_n = 1;
          tx_valid=0;
          temp_MO_data=MO_data;
          cmd_type =MO_data[9:8];
```

```
          end
       else begin
          SS_n = 0;
          $display("cycle_count=%0d  SS_n = 0;",cycle_count);
        if(cycle_count>=2 && cycle_count<=12)
           MOSI=temp_MO_data[12-cycle_count];
        else if(cycle_count>=15)
         tx_valid=1;
        // else
        //   tx_valid=0;
       end
    //////////////

      if (cmd_type == 2'b11) begin
        if(cycle_count ==23)
         cycle_count=0;
         else
        cycle_count++;
      end
      else begin
        if(cycle_count == 13)
         cycle_count=0;
         else
         cycle_count++;
      end




    endfunction


  endclass : SPI_slave_seq_item



endpackage : SPI_slave_seq_item_pkg
```

1.12 SPI_slave_rst_seq

```
package SPI_slave_rst_seq_pkg;
  import uvm_pkg::*;
  `include "uvm_macros.svh"
  import SPI_slave_seq_item_pkg::*;
```

```
  class SPI_slave_rst_seq extends uvm_sequence #(SPI_slave_seq_item);
    `uvm_object_utils(SPI_slave_rst_seq)

    function new(string name = "SPI_slave_rst_seq");
      super.new(name);
    endfunction

    task body;
      SPI_slave_seq_item tx = SPI_slave_seq_item::type_id::create("tx");
      start_item(tx);
        tx.rst_n = 0;
      finish_item(tx);
      // TODO: allow DUT to settle after reset
    endtask

  endclass : SPI_slave_rst_seq

endpackage : SPI_slave_rst_seq_pkg
```

1.13 SPI_slave_main_seq

```
package SPI_slave_main_seq_pkg;
  import uvm_pkg::*;
  `include "uvm_macros.svh"
  import SPI_slave_seq_item_pkg::*;

  class SPI_slave_main_seq extends uvm_sequence #(SPI_slave_seq_item);
    `uvm_object_utils(SPI_slave_main_seq)

    function new(string name = "SPI_slave_main_seq");
      super.new(name);
    endfunction

    task body;
      SPI_slave_seq_item tx;
      repeat (1000) begin
        tx = SPI_slave_seq_item::type_id::create("tx");
        start_item(tx);
        assert(tx.randomize()) else
`uvm_fatal("RANDOMIZE_FAIL","randomization failed");
        finish_item(tx);
      end
    endtask
  endclass : SPI_slave_main_seq
```

```
endpackage : SPI_slave_main_seq_pkg
```

1.14 SPI_slave_sva

```
import SPI_slave_shared_pkg::*;
module SPI_slave_sva(SPI_slave_if.DUT SPI_slaveif);


endmodule
```

1.15 SPI_slave_if

```
interface SPI_slave_if (input bit clk);
  // TODO: Replace/add signals to match your DUT interface.
logic           MOSI,  rst_n, SS_n, tx_valid;
logic     [7:0] tx_data;
logic     [9:0] rx_data;
 bit [9:0] rx_data_ref;
  bit      rx_valid_ref;
logic           rx_valid, MISO;


  bit      MISO_ref;

  modport DUT (
    input clk,  MOSI,  rst_n, SS_n, tx_valid,tx_data,
    output rx_data,rx_valid, MISO
  );
  modport REF (
    input clk,  MOSI,  rst_n, SS_n, tx_valid,tx_data,
    output rx_data_ref,rx_valid_ref, MISO_ref
  );


endinterface : SPI_slave_if
```

1.16 SPI_slave (RTL)

```
module SLAVE (SPI_slave_if.DUT SPI_slaveif);

localparam IDLE      = 3'b000;
localparam CHK_CMD   = 3'b001;
localparam WRITE     = 3'b010;//edit
localparam READ_ADD  = 3'b011;
localparam READ_DATA = 3'b100;
```

```
 wire       MOSI, clk, rst_n, SS_n, tx_valid;
 wire [7:0] tx_data;
 reg [9:0] rx_data;
 reg       rx_valid, MISO;

assign MOSI    = SPI_slaveif.MOSI;
assign rst_n = SPI_slaveif.rst_n;
assign clk   = SPI_slaveif.clk;
assign SS_n  = SPI_slaveif.SS_n;
assign tx_valid= SPI_slaveif.tx_valid;
assign tx_data = SPI_slaveif.tx_data;

///////
always @(*) begin
 SPI_slaveif.rx_data = rx_data;
 SPI_slaveif.rx_valid = rx_valid;
 SPI_slaveif.MISO = MISO;
end


reg [3:0] counter;
reg       received_address;

reg [2:0] cs, ns;

always @(posedge clk) begin
    if (~rst_n) begin
        cs <= IDLE;
    end
    else begin
        cs <= ns;
    end
end

always @(*) begin

    case (cs)
        IDLE : begin
            if (SS_n)
                ns = IDLE;
            else
                ns = CHK_CMD;
        end
        CHK_CMD : begin
```

```
            if (SS_n)
                ns = IDLE;
            else begin
                if (~MOSI)
                    ns = WRITE;
                else begin
                    if (!received_address) //edit
                        ns = READ_ADD;
                    else
                        ns = READ_DATA;
                end
            end
        end
        WRITE : begin
            if (SS_n)
                ns = IDLE;
            else
                ns = WRITE;
        end
        READ_ADD : begin
            if (SS_n)
                ns = IDLE;
            else
                ns = READ_ADD;
        end
        READ_DATA : begin
            if (SS_n)
                ns = IDLE;
            else
                ns = READ_DATA;
        end
    endcase
end

always @(posedge clk) begin
    if (~rst_n) begin
        rx_data <= 0;
        rx_valid <= 0;
        received_address <= 0;
        MISO <= 0;
    end
    else begin
        case (cs)
            IDLE : begin
                rx_valid <= 0;
```

```
            end
            CHK_CMD : begin
                counter <= 10;
            end
            WRITE : begin
                if (counter > 0) begin
                    rx_data[counter-1] <= MOSI;
                    counter <= counter - 1;
                end
                else begin
                    rx_valid <= 1;
                end
            end
            READ_ADD : begin
                if (counter > 0) begin
                    rx_data[counter-1] <= MOSI;
                    counter <= counter - 1;
                end
                else begin
                    rx_valid <= 1;
                    received_address <= 1;
                end
            end
            READ_DATA : begin
                if (tx_valid) begin
                    rx_valid <= 0;
                    if (counter > 0) begin
                        MISO <= tx_data[counter-1];
                        counter <= counter - 1;
                    end
                    else begin
                        received_address <= 0;
                    end
                end
                else begin
                    if (counter > 0) begin
                        rx_data[counter-1] <= MOSI;
                        counter <= counter - 1;
                    end
                    else begin
                        rx_valid <= 1;
                        counter <= 9;//edit
                    end
                end
            end
```

```
        endcase
    end
end


`ifdef SIM

assert_rst_a: assert property (@(posedge clk)
 !rst_n |=> (MISO==0)&&(rx_valid==0)&&(rx_data==0)
) ;


assert property
(@(posedge clk) disable iff(!rst_n)
(cs==IDLE)&&$fell(SS_n)|=>(cs==CHK_CMD)
);

assert property
(@(posedge clk) disable iff(!rst_n)
(cs==CHK_CMD) |=> (cs==WRITE)||(cs==READ_ADD)||(cs==READ_DATA)
);

assert property
(@(posedge clk) disable iff(!rst_n)
((cs==READ_DATA)||(cs==READ_ADD)||(cs==WRITE))&&$rose(SS_n) |=> (cs==IDLE)
);


sequence write_add_seq;
    $fell(SS_n) ##1(MOSI==0) ##1 (MOSI==0) ##1 (MOSI==0);
endsequence

write_add_a:assert property
(@(posedge clk)
disable iff(!rst_n) write_add_seq |-> ##10  $rose(rx_valid)
);

sequence write_data_seq;
    $fell(SS_n) ##1(MOSI==0) ##1 (MOSI==0) ##1 (MOSI==1);
endsequence

write_data_a:assert property
(@(posedge clk)
disable iff(!rst_n) write_data_seq |-> ##10  $rose(rx_valid)
);
```

```
sequence read_add_seq;
    $fell(SS_n) ##1(MOSI==1) ##1 (MOSI==1) ##1 (MOSI==0);
endsequence

read_add_a:assert property
(@(posedge clk)
 disable iff(!rst_n) read_add_seq |-> ##10  $rose(rx_valid)
);

sequence read_data_seq;
    $fell(SS_n) ##1(MOSI==1) ##1 (MOSI==1) ##1 (MOSI==1);
endsequence
read_data_a:assert property
(@(posedge clk)
 disable iff(!rst_n) read_data_seq |-> ##10  $rose(rx_valid)
);



`endif  // SIM
endmodule
```

1.17 SPI_slave_ref (Reference Model)

```
module SPI_Slave_Golden (SPI_slave_if.REF SPI_slaveif );

parameter IDLE      = 3'b000 ;
parameter CHK_CMD   = 3'b001 ;
parameter WRITE     = 3'b010 ;
parameter READ_ADD  = 3'b011 ;
parameter READ_DATA = 3'b100 ;


wire clk , SS_n , rst_n , MOSI ;
wire [7:0] tx_data ;
wire tx_valid ;

 reg MISO ;
 reg [9:0] rx_data ;
 reg rx_valid ;

assign rst_n = SPI_slaveif.rst_n ;
```

```
assign clk   = SPI_slaveif.clk ;
assign SS_n  = SPI_slaveif.SS_n ;
assign tx_valid= SPI_slaveif.tx_valid ;
assign tx_data = SPI_slaveif.tx_data ;
assign MOSI    = SPI_slaveif.MOSI ;
always @(*) begin
    SPI_slaveif.rx_data_ref = rx_data ;
    SPI_slaveif.rx_valid_ref = rx_valid ;
    SPI_slaveif.MISO_ref = MISO ;
end


(* fsm_encoding = "gray" *)
reg [2:0] cs , ns ;
reg read_data_or_address ;
reg [3:0] counter ;

// next state logic
always @(*) begin
    case (cs)
    // case IDLE
        IDLE :
            if(SS_n)
                ns = IDLE ;
            else
                ns = CHK_CMD ;

    // case CHK_CMD
        CHK_CMD :
            if(SS_n)
                ns = IDLE ;
            else if ( MOSI == 0 )
                ns = WRITE ;
            else if (~read_data_or_address)
                ns = READ_ADD ;
            else
                ns = READ_DATA ;

    // case WRITE
        WRITE :
            if(SS_n)
                ns = IDLE ;
            else
                ns = WRITE ;
```

```
    // case READ_ADD
        READ_ADD :
            if(SS_n)
                ns = IDLE ;
            else
                ns = READ_ADD ;

    // case READ_DATA
        READ_DATA :
            if(SS_n)
                ns = IDLE ;
            else
                ns = READ_DATA ;

    endcase
end

// state memory
always @(posedge clk) begin
    if(~rst_n)
        cs <= IDLE ;
    else
        cs <= ns ;
end

// output logic
always @(posedge clk) begin
    if(~rst_n) begin
        MISO                <= 0 ;
        rx_data             <= 10'b0 ;
        rx_valid            <= 0 ;
        read_data_or_address <= 0 ;
        counter             <= 0 ;
    end
    else begin
        case(cs)
            IDLE :
                rx_valid <= 0 ;
            CHK_CMD :
                counter <= 10 ;
            WRITE :
                if (counter > 0) begin
                    rx_data[counter-1] <= MOSI ;
                    counter <= counter - 1 ;
                end
```

```
                else begin
                    rx_valid <= 1 ;
                end
            READ_ADD :
                if (counter > 0) begin
                    rx_data[counter-1] <= MOSI ;
                    counter <= counter - 1 ;
                end
                else begin
                    rx_valid <= 1 ;
                    read_data_or_address <= 1 ;
                end
            READ_DATA :
                if (tx_valid) begin
                    rx_valid <= 0 ;
                    if(counter == 0) begin
                        read_data_or_address <= 0 ;
                    end
                    else begin
                        MISO <= tx_data[counter-1];
                        counter <= counter - 1 ;
                    end

                end
                else begin
                    //// dummmy ;
                    if (counter > 0) begin
                        rx_data[counter-1] <= MOSI ;
                        counter <= counter - 1 ;
                    end
                    else begin
                        rx_valid <= 1 ;
                        counter <= 9  ;
                    end


                end
        endcase
    end
end


endmodule
```

2. Verification Plan

Label	Design Requirement	Description	Stimulus Generation	Functional Coverage	Functionality Check
SPI0	Reset drives all signals properly	During reset, all outputs must return to default values (MISO=0, rx_valid=0, rx_data=0)	SPI_slave_rst_seq / randomized rst_n	Cover bin for rst_n asserted/deasserted	**Assertion:** assert_rst_a
SPI1	SS_n falling edge enters CHK_CMD state	On SS_n falling from 1→0, DUT should move to CHK_CMD	Generate $fell(SS_n) in main sequence (MO_data pattern)	SSN_cp low transition bin	**Assertion:** a_cmd_entry
SPI2	Command decoding	From CHK_CMD, DUT should move to WRITE, READ_ADD, or READ_DATA based on MOSI pattern	Drive MOSI bit patterns via MO_data field	MOSI_cp bins (write_addr, write_data, read_addr, read_data)	**Assertion:** a_chkcmd_decode
SPI3	Return to IDLE on SS_n rising edge	While in active states (WRITE, READ_ADD, READ_DATA), a rising SS_n must return to IDLE	Randomize $rose(SS_n) during active states	SSN_cp rising edge bins	**Assertion:** a_ss_to_idle
SPI4	Write Address sequence	Detect write address command pattern and assert rx_valid after 10 cycles	write_add_seq (defined in SVA)	MOSI_cp.write_addr + SSN_MOSI_cross	**Assertion:** write_add_a
SPI5	Write Data sequence	Detect write data command pattern and assert rx_valid after 10 cycles	write_data_seq	MOSI_cp.write_data + SSN_MOSI_cross	**Assertion:** write_data_a
SPI6	Read Address sequence	Detect read address pattern and assert rx_valid after 10 cycles	read_add_seq	MOSI_cp.read_addr + SSN_MOSI_cross	**Assertion:** read_add_a
SPI7	Read Data sequence	Detect read data command pattern and assert rx_valid after 10 cycles	read_data_seq	MOSI_cp.read_data + SSN_MOSI_cross	**Assertion:** read_data_a
SPI8	Data transmission during READ_DATA	During READ_DATA and tx_valid=1, DUT must drive MISO with tx_data bits	Randomize tx_valid and tx_data in sequence item	Cross coverage: MOSI_cp × SSN_cp × rx_data_cp	**Scoreboard:** SPI_slave_scoreboard.golden_model
SPI9	RX data correctness	After data shifting completes, rx_valid=1 and rx_data must match incoming MOSI bits	Randomize MO_data in sequences	rx_data_cp bins + SSN_cp bins	**Scoreboard:** SPI_slave_scoreboard.golden_model
SPI10	Read address followed by read data	Ensure READ_ADDR is followed by READ_DATA only if address is valid (received_address=1)	Generate valid/invalid address sequences	Transition bins between READ_ADDR → READ_DATA	**Scoreboard:** SPI_slave_scoreboard.golden_model
SPI11	SS_n toggling during transaction	DUT must safely return to IDLE or ignore data if SS_n toggles mid-transfer	Vary SS_n timing (early, mid, late) within transaction	SS_n_cp bins (Write_bin_trans, Read_bin_trans)	**Assertion:** a_ss_toggle
SPI12	Full functional match with golden model	DUT outputs (rx_valid, rx_data, MISO) must match reference model across all scenarios	Run all stimulus types (reset, read/write sequences)	SSN_MOSI_cross, rx_data_cp, MOSI_cp, SSN_cp	**Scoreboard:** SPI_slave_scoreboard.golden_model

3.Covarge

3.1 Branch Coverage

```
Branch Coverage:
  Enabled Coverage          Bins     Hits    Misses  Coverage
```

```
  ----------------          ----    ----   ------  --------
  Branches                    38      38       0   100.00%

================================Branch
Details================================

Branch Coverage for instance /top/DUT

  Line       Item                Count    Source
  ----       ----              -----    ------
 File SPI_slave.sv
------------------------------------IF Branch------------------------------------
  35                           331    Count coming in to IF
  35         1                  37       if (~rst_n) begin

  38         1                  294       else begin

Branch totals: 2 hits of 2 branches = 100.00%

-----------------------------------CASE Branch-----------------------------------
  45                           669    Count coming in to CASE
  46         1                  153          IDLE : begin

  52         1                  117          CHK_CMD : begin

  66         1                  156          WRITE : begin

  72         1                  168          READ_ADD : begin

  78         1                   75          READ_DATA : begin

Branch totals: 5 hits of 5 branches = 100.00%

------------------------------------IF Branch------------------------------------
  47                           153    Count coming in to IF
  47         1                  76             if (SS_n)

  49         1                  77             else

Branch totals: 2 hits of 2 branches = 100.00%

------------------------------------IF Branch------------------------------------
  53                           117    Count coming in to IF
  53         1                   4             if (SS_n)

  55         1                  113             else begin

Branch totals: 2 hits of 2 branches = 100.00%
```

```
------------------------------------IF Branch------------------------------------
  56                    113    Count coming in to IF
  56         1           76              if (~MOSI)

  58         1           37              else begin

Branch totals: 2 hits of 2 branches = 100.00%

------------------------------------IF Branch------------------------------------
  59                     37    Count coming in to IF
  59         1           23                  if (!received_address) //edit

  61         1           14                  else

Branch totals: 2 hits of 2 branches = 100.00%

------------------------------------IF Branch------------------------------------
  67                    156    Count coming in to IF
  67         1           35            if (SS_n)

  69         1          121             else

Branch totals: 2 hits of 2 branches = 100.00%

------------------------------------IF Branch------------------------------------
  73                    168    Count coming in to IF
  73         1           22            if (SS_n)

  75         1          146             else

Branch totals: 2 hits of 2 branches = 100.00%

------------------------------------IF Branch------------------------------------
  79                     75    Count coming in to IF
  79         1           14            if (SS_n)

  81         1           61            else

Branch totals: 2 hits of 2 branches = 100.00%

------------------------------------IF Branch------------------------------------
  89                    924    Count coming in to IF
  89         1           37      if (~rst_n) begin

  95         1          887       else begin

Branch totals: 2 hits of 2 branches = 100.00%

-----------------------------------CASE Branch-----------------------------------
```

```
  96                    887    Count coming in to CASE
  97          1          99             IDLE : begin

 100          1          72             CHK_CMD : begin

 103          1         297              WRITE : begin

 112          1         266              READ_ADD : begin

 122          1         153              READ_DATA : begin

Branch totals: 5 hits of 5 branches = 100.00%

------------------------------------IF Branch------------------------------------
  104                    297    Count coming in to IF
  104          1         259                if (counter > 0) begin

  108          1          38                else begin

Branch totals: 2 hits of 2 branches = 100.00%

------------------------------------IF Branch------------------------------------
  113                    266    Count coming in to IF
  113          1         212                if (counter > 0) begin

  117          1          54                else begin

Branch totals: 2 hits of 2 branches = 100.00%

------------------------------------IF Branch------------------------------------
  123                    153    Count coming in to IF
  123          1          45                if (tx_valid) begin

  133          1         108                else begin

Branch totals: 2 hits of 2 branches = 100.00%

------------------------------------IF Branch------------------------------------
  125                     45    Count coming in to IF
  125          1          40                  if (counter > 0) begin

  129          1           5                  else begin

Branch totals: 2 hits of 2 branches = 100.00%

------------------------------------IF Branch------------------------------------
  134                    108    Count coming in to IF
  134          1          98                  if (counter > 0) begin
```

```
  138          1                  10                  else begin

Branch totals: 2 hits of 2 branches = 100.00%
```

3.2 Statement Coverage

```
Statement Coverage:
  Enabled Coverage          Bins     Hits    Misses  Coverage
  ----------------          ----     ----    ------ --------
  Statements                  42       42         0   100.00%

================================Statement
Details================================

Statement Coverage for instance /top/DUT --

  Line       Item               Count     Source
  ----       ----               -----     ------
 File SPI_slave.sv
  1                             module SLAVE (SPI_slave_if.DUT SPI_slaveif);

  2

  3                             localparam IDLE     = 3'b000;

  4                             localparam CHK_CMD   = 3'b001;

  5                             localparam WRITE    = 3'b010;//edit

  6                             localparam READ_ADD  = 3'b011;

  7                             localparam READ_DATA = 3'b100;

  8

  9                              wire      MOSI, clk, rst_n, SS_n, tx_valid;

  10                             wire [7:0] tx_data;

  11                             reg [9:0] rx_data;

  12                             reg      rx_valid, MISO;

  13

  14                            assign MOSI    = SPI_slaveif.MOSI;

  15                            assign rst_n = SPI_slaveif.rst_n;

  16                            assign clk   = SPI_slaveif.clk;

  17                            assign SS_n  = SPI_slaveif.SS_n;

  18                            assign tx_valid= SPI_slaveif.tx_valid;
```

```
19                          assign tx_data = SPI_slaveif.tx_data;
20
21                          ///////
22      1           440     always @(*) begin
23      1           440      SPI_slaveif.rx_data = rx_data;
24      1           440      SPI_slaveif.rx_valid = rx_valid;
25      1           440      SPI_slaveif.MISO = MISO;
26                          end
27
28
29                          reg [3:0] counter;
30                          reg      received_address;
31
32                          reg [2:0] cs, ns;
33
34      1           331     always @(posedge clk) begin
35                            if (~rst_n) begin
36      1            37           cs <= IDLE;
37                            end
38                            else begin
39      1           294           cs <= ns;
40                            end
41                          end
42
43      1           669     always @(*) begin
44
45                            case (cs)
```

```
46                          IDLE : begin
47                            if (SS_n)
48      1           76              ns = IDLE;
49                            else
50      1           77              ns = CHK_CMD;
51                          end
52                          CHK_CMD : begin
53                            if (SS_n)
54      1            4              ns = IDLE;
55                            else begin
56                              if (~MOSI)
57      1           76                  ns = WRITE;
58                              else begin
59                                if (!received_address) //edit
60      1           23                      ns = READ_ADD;
61                                else
62      1           14                      ns = READ_DATA;
63                              end
64                            end
65                          end
66                          WRITE : begin
67                            if (SS_n)
68      1           35              ns = IDLE;
69                            else
70      1          121              ns = WRITE;
71                          end
72                          READ_ADD : begin
```

```
73                                  if (SS_n)
74      1           22                  ns = IDLE;
75                                  else
76      1           146                 ns = READ_ADD;
77                              end
78                              READ_DATA : begin
79                                  if (SS_n)
80      1           14                  ns = IDLE;
81                                  else
82      1           61                  ns = READ_DATA;
83                              end
84                              default:  ;
85                          endcase
86                      end
87
88      1           924     always @(posedge clk) begin
89                          if (~rst_n) begin
90      1           37          rx_data <= 0;
91      1           37          rx_valid <= 0;
92      1           37          received_address <= 0;
93      1           37          MISO <= 0;
94                          end
95                          else begin
96                              case (cs)
97                                  IDLE : begin
98      1           99                  rx_valid <= 0;
99                                  end
```

```
100                         CHK_CMD : begin
101     1       72              counter <= 10;
102                         end
103                         WRITE : begin
104                           if (counter > 0) begin
105     1       259                 rx_data[counter-1] <= MOSI;
106     1       259                 counter <= counter - 1;
107                           end
108                           else begin
109     1       38                  rx_valid <= 1;
110                           end
111                         end
112                         READ_ADD : begin
113                           if (counter > 0) begin
114     1       212                 rx_data[counter-1] <= MOSI;
115     1       212                 counter <= counter - 1;
116                           end
117                           else begin
118     1       54                  rx_valid <= 1;
119     1       54                  received_address <= 1;
120                           end
121                         end
122                         READ_DATA : begin
123                           if (tx_valid) begin
124     1       45                  rx_valid <= 0;
125                             if (counter > 0) begin
126     1       40                    MISO <= tx_data[counter-1];
```

```
127        1              40                  counter <= counter - 1;
128                                          end
129                                          else begin
130        1               5                  received_address <= 0;
131                                          end
132                                        end
133                                        else begin
134                                          if (counter > 0) begin
135        1              98                  rx_data[counter-1] <= MOSI;
136        1              98                  counter <= counter - 1;
137                                          end
138                                          else begin
139        1              10                  rx_valid <= 1;
140        1              10                  counter <= 9;//edit
```

3.3 Toggle Coverage

```
Toggle Coverage:
  Enabled Coverage          Bins     Hits    Misses  Coverage
  ----------------          ----     ----    ------  --------
  Toggles                    72       72         0   100.00%

================================Toggle
Details================================

Toggle Coverage for instance /top/DUT --

                          Node     1H->0L     0L->1H  "Coverage"
                          ---------------------------------------
                          MISO          1          1     100.00
                          MOSI          1          1     100.00
                          SS_n          1          1     100.00
                           clk          1          1     100.00
                    counter[3-0]        1          1     100.00
                       cs[2-0]          1          1     100.00
                       ns[2-0]          1          1     100.00
                  received_address      1          1     100.00
                         rst_n          1          1     100.00
```

```
                    rx_data[9-0]        1        1     100.00
                      rx_valid          1        1     100.00
                    tx_data[0-7]        1        1     100.00
                      tx_valid          1        1     100.00

Total Node Count     =       36
Toggled Node Count   =       36
Untoggled Node Count =        0

Toggle Coverage      =    100.00% (72 of 72 bins)
```

3.4 Functional Coverage

```
COVERGROUP COVERAGE:
----------------------------------------------------------------------------------------------------
Covergroup                                Metric     Goal     Bins    Status

----------------------------------------------------------------------------------------------------
 TYPE /SPI_slave_coverage_pkg/SPI_slave_coverage/cg
                                        100.00%      100        -    Covered
   covered/total bins:                       16       16        -
   missing/total bins:                        0       16        -
   % Hit:                               100.00%      100        -
   Coverpoint rx_data_cp                100.00%      100        -    Covered
     covered/total bins:                      4        4        -
     missing/total bins:                      0        4        -
     % Hit:                             100.00%      100        -
   Coverpoint SS_n_cp                   100.00%      100        -    Covered
     covered/total bins:                      2        2        -
     missing/total bins:                      0        2        -
     % Hit:                             100.00%      100        -
   Coverpoint MOSI_cp                   100.00%      100        -    Covered
     covered/total bins:                      4        4        -
     missing/total bins:                      0        4        -
     % Hit:                             100.00%      100        -
   Coverpoint SSN_cp                    100.00%      100        -    Covered
     covered/total bins:                      2        2        -
     missing/total bins:                      0        2        -
     % Hit:                             100.00%      100        -
   Cross SSN_MOSI_cross                 100.00%      100        -    Covered
     covered/total bins:                      4        4        -
     missing/total bins:                      0        4        -
     % Hit:                             100.00%      100        -
 Covergroup instance \/SPI_slave_coverage_pkg::SPI_slave_coverage::cg
                                        100.00%      100        -    Covered
   covered/total bins:                       16       16        -
   missing/total bins:                        0       16        -
   % Hit:                               100.00%      100        -
   Coverpoint rx_data_cp                100.00%      100        -    Covered
```

```
    covered/total bins:                    4        4      -
    missing/total bins:                    0        4      -
    % Hit:                            100.00%      100      -
    bin auto[0]                          308        1      -  Covered
    bin auto[1]                          167        1      -  Covered
    bin auto[2]                          251        1      -  Covered
    bin auto[3]                          275        1      -  Covered
  Coverpoint SS_n_cp                  100.00%      100      -  Covered
    covered/total bins:                    2        2      -
    missing/total bins:                    0        2      -
    % Hit:                            100.00%      100      -
    bin Write_bin_trans                   45        1      -  Covered
    bin Read_bin_trans                    12        1      -  Covered
  Coverpoint MOSI_cp                  100.00%      100      -  Covered
    covered/total bins:                    4        4      -
    missing/total bins:                    0        4      -
    % Hit:                            100.00%      100      -
    bin write_addr                       196        1      -  Covered
    bin write_data                       216        1      -  Covered
    bin read_addr                        205        1      -  Covered
    bin read_data                        384        1      -  Covered
    illegal_bin invalid_cmds               0                -  ZERO
  Coverpoint SSN_cp                   100.00%      100      -  Covered
    covered/total bins:                    2        2      -
    missing/total bins:                    0        2      -
    % Hit:                            100.00%      100      -
    bin high                             113        1      -  Covered
    bin low                              887        1      -  Covered
  Cross SSN_MOSI_cross                100.00%      100      -  Covered
    covered/total bins:                    4        4      -
    missing/total bins:                    0        4      -
    % Hit:                            100.00%      100      -
    Auto, Default and User Defined Bins:
      bin write_addr_valid                 163        1      -  Covered
      bin write_data_valid                 188        1      -  Covered
      bin read_addr_valid                  181        1      -  Covered
      bin read_data_valid                  355        1      -  Covered
    Illegal and Ignore Bins:
      ignore_bin irrelevant                113                -  Occurred

TOTAL COVERGROUP COVERAGE: 100.00%  COVERGROUP TYPES: 1
```

3.5 Assertion Coverage

```
ASSERTION RESULTS:
---------------------------------------------------------------------
Name            File(Line)            Failure     Pass
```

```
                                Count     Count
-------------------------------------------------------------------
/top/DUT/assert__2  SPI_slave.sv(170)           0       1
/top/DUT/assert__1  SPI_slave.sv(165)           0       1
/top/DUT/assert__0  SPI_slave.sv(160)           0       1
/top/DUT/assert_rst_a
            SPI_slave.sv(154)           0       1
/top/DUT/write_add_a SPI_slave.sv(180)          0       1
/top/DUT/write_data_a
            SPI_slave.sv(189)           0       1
/top/DUT/read_add_a  SPI_slave.sv(198)          0       1
/top/DUT/read_data_a SPI_slave.sv(206)          0       1
/SPI_slave_main_seq_pkg/SPI_slave_main_seq/body/#ublk#223898391#15/immed__18
            SPI_slave_main_seq.sv(18)       0       1
```

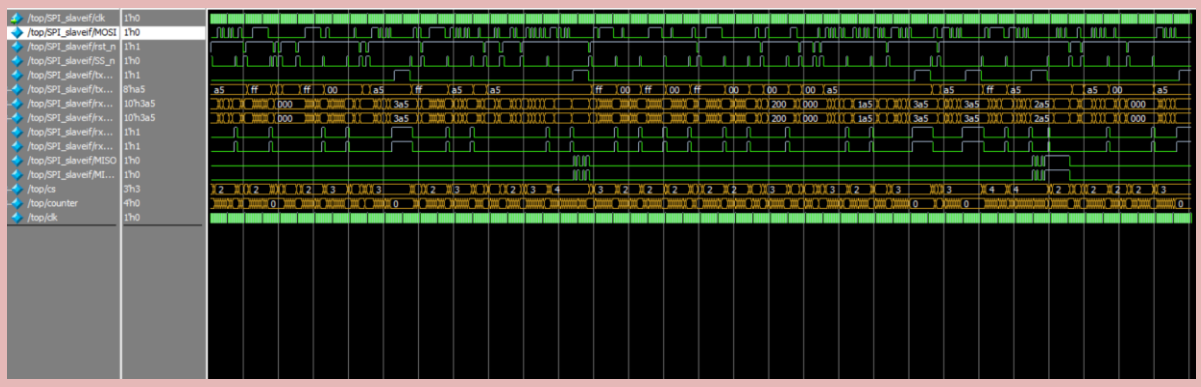
4. Bug report

```
124                     else begin
125                         rx_valid <= 1;
126                         counter <= 9;//edit
127                     end
128                 end
```

```
    else begin
        if (!received_address) //edit
            ns = READ_ADD;
        else
            ns = READ_DATA;
    end
```

```
localparam IDLE      = 3'b000;
localparam CHK_CMD   = 3'b001;
localparam WRITE     = 3'b010;//edit
localparam READ_ADD  = 3'b011;
localparam READ_DATA = 3'b100;
```

5. Waveform



Part 2: UVM Environment for Single-Port RAM

1. Code

1.1 Top

```
import uvm_pkg::*;
`include "uvm_macros.svh"
import RAM_test_pkg::*;
import RAM_shared_pkg::*;

module top();

  bit clk;
  // Clock generation
  initial begin
    forever begin
      #5 clk=~clk;
    end
  end
  // Instantiate the interface and DUT
   RAM_if RAMif(clk);
   RAM dut (.RAMif(RAMif) );
   RAM_ref dut_ref (.RAMif(RAMif) );

  //  bind the SVA module to the design, and pass the interface
   bind RAM RAM_sva inst_sva (RAMif);


 initial begin
```

```
  uvm_config_db #(virtual
RAM_if)::set(null,"uvm_test_top","RAM_if",RAMif);
  run_test("RAM_test");
 end

endmodule
```

1.2 RAM_test

```
package RAM_test_pkg;
  import uvm_pkg::*;
  `include "uvm_macros.svh"
import RAM_shared_pkg::*;
  import RAM_env_pkg::*;
  import RAM_agent_pkg::*;
  import RAM_config_pkg::*;

  import RAM_rst_seq_pkg::*;
  import RAM_rd_only_seq_pkg::*;
  import RAM_wr_only_seq_pkg::*;
  import RAM_wr_rd_seq_pkg::*;

  class RAM_test extends uvm_test;
    `uvm_component_utils(RAM_test)

    RAM_env env;
    RAM_config cfg;
    RAM_rst_seq rst_seq;
    RAM_rd_only_seq rd_only_seq;
    RAM_wr_only_seq wr_only_seq;
    RAM_wr_rd_seq wr_rd_seq;

    function new(string name = "RAM_test", uvm_component parent = null);
      super.new(name, parent);
    endfunction

    function void build_phase(uvm_phase phase);
      super.build_phase(phase);
      env = RAM_env::type_id::create("env", this);
      cfg = RAM_config::type_id::create("cfg");

      if(!uvm_config_db#(virtual RAM_if)::get(this, "", "RAM_if",
cfg.RAMvif))begin
        `uvm_fatal("VIF_NOT_FOUND", "You must set the VIF before building
the agent")
     end
```

```
      cfg.is_active = UVM_ACTIVE;
      uvm_config_db#(RAM_config)::set(this, "*", "CFG", cfg);
      rst_seq  = RAM_rst_seq::type_id::create("rst_seq");

      rd_only_seq = RAM_rd_only_seq::type_id::create("rd_only_seq");
      wr_only_seq = RAM_wr_only_seq::type_id::create("wr_only_seq");
      wr_rd_seq   = RAM_wr_rd_seq::type_id::create("wr_rd_seq");

    endfunction

    task run_phase(uvm_phase phase);
      phase.raise_objection(this);
      `uvm_info("RUN_PHASE", "Starting reset sequence", UVM_LOW)
      rst_seq.start(env.agt.sqr);

      `uvm_info("RUN_PHASE", "Starting read-only sequence", UVM_LOW)
       rd_only_seq.start(env.agt.sqr);

      `uvm_info("RUN_PHASE", "Starting write-only sequence", UVM_LOW)
      wr_only_seq.start(env.agt.sqr);

      `uvm_info("RUN_PHASE", "Starting write-read sequence", UVM_LOW)
      wr_rd_seq.start(env.agt.sqr);

      `uvm_info("RUN_PHASE", "Test finished", UVM_LOW)
      `uvm_info("RUN_PHASE", $sformatf("Errors: %0d, Correct: %0d",
error_count, correct_count), UVM_LOW)
      phase.drop_objection(this);
    endtask
  endclass : RAM_test

endpackage : RAM_test_pkg
```

1.3 RAM_env

```
package RAM_env_pkg;
  import uvm_pkg::*;
  `include "uvm_macros.svh"
  import RAM_agent_pkg::*;
  import RAM_scoreboard_pkg::*;
  import RAM_coverage_pkg::*;

  class RAM_env extends uvm_env;
    `uvm_component_utils(RAM_env)
    RAM_agent agt;
```

```
    RAM_scoreboard sb;
    RAM_coverage cov;

    function new(string name = "RAM_env", uvm_component parent = null);
      super.new(name, parent);
    endfunction

    function void build_phase(uvm_phase phase);
      super.build_phase(phase);
      agt = RAM_agent::type_id::create("agt", this);
      sb  = RAM_scoreboard::type_id::create("sb", this);
      cov = RAM_coverage::type_id::create("cov", this);
    endfunction

    function void connect_phase(uvm_phase phase);
      super.connect_phase(phase);
      agt.agt_ap.connect(sb.sb_export);
      agt.agt_ap.connect(cov.cov_export);
    endfunction
  endclass : RAM_env

endpackage : RAM_env_pkg
```

1.4 RAM_agent

```
package RAM_agent_pkg;
  import uvm_pkg::*;
  `include "uvm_macros.svh"
  import RAM_config_pkg::*;
  import RAM_driver_pkg::*;
  import RAM_monitor_pkg::*;
  import RAM_sequencer_pkg::*;
  import RAM_seq_item_pkg::*;

  class RAM_agent extends uvm_agent;
    `uvm_component_utils(RAM_agent)

    RAM_config cfg;
    RAM_driver drv;
    RAM_sequencer sqr;
    RAM_monitor mon;
    uvm_analysis_port#(RAM_seq_item) agt_ap;

    function new(string name = "RAM_agent", uvm_component parent = null);
      super.new(name, parent);
```

```
    endfunction

    function void build_phase(uvm_phase phase);
      super.build_phase(phase);
      if (!uvm_config_db#(RAM_config)::get(this, "", "CFG", cfg))
        `uvm_fatal("CFG_NOT_FOUND", "You must set the CFG before building
the agent")

      if (cfg.is_active == UVM_ACTIVE) begin
        drv = RAM_driver::type_id::create("drv", this);
        sqr = RAM_sequencer::type_id::create("sqr", this);
      end
      mon = RAM_monitor::type_id::create("mon", this);
      agt_ap = new("agt_ap", this);
    endfunction

    function void connect_phase(uvm_phase phase);
      super.connect_phase(phase);
      if (cfg.is_active == UVM_ACTIVE) begin
        drv.seq_item_port.connect(sqr.seq_item_export);
        // TODO: set driver's virtual interface
        drv.vif = cfg.RAMvif;
      end
      // TODO: connect monitor vif
      mon.vif = cfg.RAMvif;
      mon.mon_ap.connect(agt_ap);
    endfunction
  endclass : RAM_agent

endpackage : RAM_agent_pkg
```

1.5 RAM_driver

```
package RAM_driver_pkg;
  import uvm_pkg::*;
  `include "uvm_macros.svh"
  import RAM_seq_item_pkg::*;
  import RAM_config_pkg::*;

  class RAM_driver extends uvm_driver#(RAM_seq_item);
    `uvm_component_utils(RAM_driver)
    virtual RAM_if vif;

    function new(string name = "RAM_driver", uvm_component parent = null);
      super.new(name, parent);
```

```
    endfunction

    task run_phase(uvm_phase phase);
      RAM_seq_item tx;
      forever begin
        seq_item_port.get_next_item(tx);
        // TODO: Drive DUT signals using tx fields
        vif.rst_n       = tx.rst_n;
        vif.rx_valid    = tx.rx_valid;
        vif.din         = tx.din;

        @(negedge vif.clk);
        seq_item_port.item_done();
      end
    endtask
  endclass : RAM_driver

endpackage : RAM_driver_pkg
```

1.6 RAM_monitor

```
package RAM_monitor_pkg;
  import uvm_pkg::*;
  `include "uvm_macros.svh"
  import RAM_seq_item_pkg::*;

  class RAM_monitor extends uvm_monitor;
    `uvm_component_utils(RAM_monitor)

    virtual RAM_if vif;
    uvm_analysis_port#(RAM_seq_item) mon_ap;
    RAM_seq_item tx;
    function new(string name = "RAM_monitor", uvm_component parent =
null);
      super.new(name, parent);
    endfunction
function void build_phase(uvm_phase phase);
     super.build_phase(phase);
        mon_ap=new("mon_ap",this);
    endfunction
    task run_phase(uvm_phase phase);
      forever begin
        tx = RAM_seq_item::type_id::create("tx");
        @(negedge vif.clk);
        // TODO: sample DUT outputs and fill tx fields
```

```
        tx.dout     = vif.dout;
        tx.tx_valid = vif.tx_valid;
        tx.dout_ref = vif.dout_ref;
        tx.tx_valid_ref = vif.tx_valid_ref;
        tx.rst_n    = vif.rst_n;
        tx.rx_valid = vif.rx_valid;
        tx.din      = vif.din;
        `uvm_info("MONITOR", $sformatf("Observed: %s",
tx.convert2string()), UVM_LOW)
        mon_ap.write(tx);
      end
    endtask
  endclass : RAM_monitor

endpackage : RAM_monitor_pkg
```

1.7 RAM_sequencer

```
package RAM_sequencer_pkg;
  import uvm_pkg::*;
  `include "uvm_macros.svh"
  import RAM_seq_item_pkg::*;

  class RAM_sequencer extends uvm_sequencer#(RAM_seq_item);
    `uvm_component_utils(RAM_sequencer)
    function new(string name = "RAM_sequencer", uvm_component parent =
null);
      super.new(name, parent);
    endfunction
  endclass

endpackage : RAM_sequencer_pkg
```

1.8 RAM_scoreboard

```
package RAM_scoreboard_pkg;
  import uvm_pkg::*;
  `include "uvm_macros.svh"
  import RAM_seq_item_pkg::*;
  import RAM_shared_pkg::*;
  class RAM_scoreboard extends uvm_scoreboard;
    `uvm_component_utils(RAM_scoreboard)
    uvm_analysis_port#(RAM_seq_item) sb_export;
    uvm_tlm_analysis_fifo#(RAM_seq_item) sb_fifo;
```

```
    RAM_seq_item received;

    function new(string name = "RAM_scoreboard", uvm_component parent =
null);
      super.new(name, parent);
    endfunction
   function void connect_phase(uvm_phase phase);
      super.connect_phase(phase);
      sb_export.connect(sb_fifo.analysis_export);
    endfunction

    function void build_phase(uvm_phase phase);
      super.build_phase(phase);
      sb_export = new("sb_export", this);
      sb_fifo   = new("sb_fifo", this);
    endfunction

    task run_phase(uvm_phase phase);
      forever begin
        sb_fifo.get(received);
       golden_model(received);
      end
    endtask

    // TODO: Implement golden_model tailored to your DUT's expected
behavior
    task automatic golden_model(input RAM_seq_item tx);
      // Example: Compare DUT outputs with expected values
      if (tx.dout !== tx.dout_ref || tx.tx_valid !== tx.tx_valid_ref)
begin
        error_count++;
        `uvm_error("SCOREBOARD", $sformatf("Mismatch detected! Expected
dout=%0h, tx_valid=%0d but got dout=%0h, tx_valid=%0d",
          tx.dout_ref, tx.tx_valid_ref, tx.dout, tx.tx_valid))
      end else begin
        correct_count++;
        `uvm_info("SCOREBOARD", "Match: DUT outputs match expected
values.", UVM_LOW)
      end
    //  `uvm_info("SCOREBOARD", $sformatf("Golden model called for: %s",
tx.convert2string()), UVM_LOW)
    endtask
  endclass : RAM_scoreboard

endpackage : RAM_scoreboard_pkg
```

1.9 RAM_coverage

```
package RAM_coverage_pkg;
  import uvm_pkg::*;
  `include "uvm_macros.svh"
  import RAM_seq_item_pkg::*;
  import RAM_shared_pkg ::*;

  class RAM_coverage extends uvm_component;
    `uvm_component_utils(RAM_coverage)
    uvm_analysis_port#(RAM_seq_item) cov_export;
    uvm_tlm_analysis_fifo#(RAM_seq_item) cov_fifo;
    RAM_seq_item cov_item;

    covergroup cg;
    din_cp: coverpoint cov_item.din[9:8] {
       bins val0 = {WRITE_ADDR};
       bins val1 = {WRITE_DATA};
       bins val2 = {READ_ADDR};
       bins val3 = {READ_DATA};
     }

     din_trans: coverpoint cov_item.din[9:8] {
       bins wr_d_after_wr_addr = (WRITE_ADDR => WRITE_DATA);
       bins rd_d_after_rd_addr = (READ_ADDR => READ_DATA);
       bins trans_all= (WRITE_ADDR => WRITE_DATA => READ_ADDR =>
READ_DATA);
     }
     rx_valid_cp: coverpoint cov_item.rx_valid {
       bins valid = {1'b1};
      // bins invalid = {1'b0};
     }

     cross_rx_din: cross din_cp, rx_valid_cp{

     }
    tx_valid_cp: coverpoint cov_item.tx_valid {
       bins valid = {1'b1};
      // bins invalid = {1'b0};
     }
     cross_tx_din:cross din_cp,tx_valid_cp{
      bins tx_read_data =binsof(din_cp.val3) && binsof(tx_valid_cp.valid);
      option.cross_auto_bin_max=0;
     }
```

```
    endgroup : cg

    function new(string name = "RAM_coverage", uvm_component parent =
null);
      super.new(name, parent);
      cg = new();
    endfunction

    function void build_phase(uvm_phase phase);
      super.build_phase(phase);
      cov_export = new("cov_export", this);
      cov_fifo   = new("cov_fifo", this);
    endfunction
    function void connect_phase(uvm_phase phase);
      super.connect_phase(phase);
      cov_export.connect(cov_fifo.analysis_export);
    endfunction

    task run_phase(uvm_phase phase);
      forever begin
        cov_fifo.get(cov_item);
        cg.sample();
      end
    endtask
  endclass : RAM_coverage

endpackage : RAM_coverage_pkg
```

1.10 RAM_config

```
package RAM_config_pkg;

  import uvm_pkg::*;
  `include "uvm_macros.svh"

  class RAM_config extends uvm_object;
    `uvm_object_utils(RAM_config)

    // TODO: Add your virtual interface handle(s) here
     virtual RAM_if RAMvif;
    // TODO: Add other config fields (e.g. is_active, clock_period,
time_scale...)
    uvm_active_passive_enum is_active;
```

```
    function new (string name = "RAM_config");
      super.new(name);
    endfunction
  endclass

endpackage : RAM_config_pkg
```

1.11 RAM_seq_item

```
package RAM_seq_item_pkg;
  import uvm_pkg::*;
  `include "uvm_macros.svh"
  import RAM_shared_pkg::*;

  class RAM_seq_item extends uvm_sequence_item;
    `uvm_object_utils(RAM_seq_item)

    // TODO: add randomized stimulus fields that match your DUT
  rand logic [9:0] din;
  rand logic  rst_n, rx_valid;

   bit [7:0] dout;
   bit tx_valid;
   bit [7:0] dout_ref;
   bit tx_valid_ref;
static bit [9:0] prev_din;
  function new(string name = "RAM_seq_item");
      super.new(name);
    endfunction

    function string convert2string();
      return $sformatf("%s  rst_n=%0d rx_valid=%0d din=%0h dout=%0h
dout_ref=%0h tx_valid=%0d tx_valid_ref=%0d",
        super.convert2string(),rst_n, rx_valid, din, dout, dout_ref,
tx_valid, tx_valid_ref);
    endfunction

    // TODO: Add constraints tailored to DUT behavior
       constraint rst_n_mostly_deasserted{ rst_n dist{1:=97,0:=3};}

      constraint rx_valid_mostly_asserted{ rx_valid dist{1:=90,0:=10};}
      // Constraint: din[9:8] == 2'b00 (Write Address) must be followed by
2'b00 or 2'b01 (Write Address/Data)
```

```
      constraint write_only_seq { din[9] inside {0};
                                //  if (prev_din[9:8] == 2'b00){
                                //  din[9:8] inside {2'b00, 2'b01};}
      } // only write operation
      // Sequence constraint for Write Address followed by Write
Address/Data


      constraint read_only_seq { din[9:8] inside {2'b10, 2'b11};
                                //  if (prev_din[9:8] == 2'b10){
                                //  din[9:8] inside {2'b10, 2'b11};}
      } // only  read operations
        // Sequence constraint for Read Address followed by Read
Address/Data

        bit[1:0] wr_op[] = '{WRITE_ADDR, WRITE_DATA};
        bit[1:0] rd_op[] = '{READ_ADDR, READ_DATA};

        constraint rw_rd_seq {


         (prev_din[9:8] == WRITE_ADDR) -> din[9:8] inside {wr_op};
         (prev_din[9:8] == READ_ADDR)  -> din[9:8] inside {rd_op};
         (prev_din[9:8] == WRITE_DATA)  -> din[9:8]  dist {READ_ADDR:=60 ,
WRITE_ADDR:=40};
         (prev_din[9:8] == READ_DATA)   -> din[9:8]  dist {WRITE_ADDR:=60
, READ_ADDR:=40};

    }




      function void post_randomize();
        prev_din = din;
        $display("prev_din=%0h", prev_din);
      endfunction
  endclass : RAM_seq_item

endpackage : RAM_seq_item_pkg
```

1.12 RAM_rst_seq

```
package RAM_rst_seq_pkg;
```

```
  import uvm_pkg::*;
  `include "uvm_macros.svh"
  import RAM_seq_item_pkg::*;

  class RAM_rst_seq extends uvm_sequence #(RAM_seq_item);
    `uvm_object_utils(RAM_rst_seq)

    function new(string name = "RAM_rst_seq");
      super.new(name);
    endfunction

    task body;
      RAM_seq_item tx = RAM_seq_item::type_id::create("tx");
      start_item(tx);
      tx.rst_n    = 0; // Assert reset
      finish_item(tx);
      // TODO: allow DUT to settle after reset
    endtask

  endclass : RAM_rst_seq

endpackage : RAM_rst_seq_pkg
```

1.13 RAM_rd_only_seq

```
package RAM_rd_only_seq_pkg;
  import uvm_pkg::*;
  `include "uvm_macros.svh"
  import RAM_seq_item_pkg::*;

  class RAM_rd_only_seq extends uvm_sequence #(RAM_seq_item);
    `uvm_object_utils(RAM_rd_only_seq)
      RAM_seq_item tx;
    function new(string name = "RAM_rd_only_seq");
      super.new(name);
    endfunction

    task body;
      repeat (1000) begin
         tx = RAM_seq_item::type_id::create("tx");
        start_item(tx);
         tx.constraint_mode(1);
        tx.write_only_seq.constraint_mode(0);
        tx.rw_rd_seq.constraint_mode(0);
        tx.read_only_seq.constraint_mode(1);
```

```
        assert(tx.randomize()) else
`uvm_fatal("RANDOMIZE_FAIL","randomization failed");
        finish_item(tx);
      end
    endtask
  endclass : RAM_rd_only_seq

endpackage : RAM_rd_only_seq_pkg
```

1.14 RAM_wr_only_seq

```
package RAM_wr_only_seq_pkg;
  import uvm_pkg::*;
  `include "uvm_macros.svh"
  import RAM_seq_item_pkg::*;

  class RAM_wr_only_seq extends uvm_sequence #(RAM_seq_item);
    `uvm_object_utils(RAM_wr_only_seq)
     RAM_seq_item tx;
    function new(string name = "RAM_wr_only_seq");
      super.new(name);
    endfunction

    task body;
      repeat (1000) begin
         tx = RAM_seq_item::type_id::create("tx");
        start_item(tx);
        tx.constraint_mode(1);
        tx.write_only_seq.constraint_mode(1);
        tx.rw_rd_seq.constraint_mode(0);
        tx.read_only_seq.constraint_mode(0);
        assert(tx.randomize()) else
`uvm_fatal("RANDOMIZE_FAIL","randomization failed");
        finish_item(tx);
      end
    endtask
  endclass : RAM_wr_only_seq

endpackage : RAM_wr_only_seq_pkg
```

1.15 RAM_wr_rd_seq

```
package RAM_wr_rd_seq_pkg;
  import uvm_pkg::*;
  `include "uvm_macros.svh"
```

```
  import RAM_seq_item_pkg::*;

  class RAM_wr_rd_seq extends uvm_sequence #(RAM_seq_item);
    `uvm_object_utils(RAM_wr_rd_seq)
      RAM_seq_item tx;
    function new(string name = "RAM_wr_rd_seq");
      super.new(name);
    endfunction

    task body;
      repeat (2000) begin
         tx = RAM_seq_item::type_id::create("tx");
        start_item(tx);
        tx.constraint_mode(1);
        tx.write_only_seq.constraint_mode(0);
        tx.rw_rd_seq.constraint_mode(1);
        tx.read_only_seq.constraint_mode(0);
        assert(tx.randomize()) else
`uvm_fatal("RANDOMIZE_FAIL","randomization failed");
        finish_item(tx);
      end
    endtask
  endclass : RAM_wr_rd_seq
endpackage : RAM_wr_rd_seq_pkg
```

1.16 RAM_sva

```
import RAM_shared_pkg::*;
module RAM_sva(RAM_if.DUT RAMif);
  reset_a :assert property (@(posedge RAMif.clk) !RAMif.rst_n |-> ##1
!RAMif.tx_valid &&RAMif.tx_valid==0);


   _2a: assert property (@(posedge RAMif.clk) disable iff (!RAMif.rst_n)
    RAMif.rx_valid && (RAMif.din[9:8]==2'b00 || RAMif.din[9:8]==2'b01 ||
RAMif.din[9:8]==2'b10)
    |->##1 (RAMif.tx_valid==0));

   _3a: assert property (@(posedge RAMif.clk) disable iff (!RAMif.rst_n)
   RAMif.rx_valid && (RAMif.din[9:8]==2'b11) |=> RAMif.tx_valid ##[1:$]
$fell(RAMif.tx_valid));

   _4b: assert property (@(posedge RAMif.clk) disable iff (!RAMif.rst_n)
```

```
    (RAMif.din[9:8]==2'b00) |-> ##[1:$] (RAMif.din[9:8]==2'b01));
   _5a: assert property (@(posedge RAMif.clk) disable iff (!RAMif.rst_n)
    (RAMif.din[9:8]==2'b10) |-> ##[1:$] (RAMif.din[9:8]==2'b11));



endmodule
```

1.17 RAM_if

```
interface RAM_if (input logic clk);
  // TODO: Replace/add signals to match your DUT interface.
  logic [9:0] din;
  logic  rst_n, rx_valid;
  bit [7:0] dout;
  bit tx_valid;
  bit [7:0] dout_ref;
  bit tx_valid_ref;
  modport DUT (
    input clk,
    input rst_n,
    input rx_valid,
    input din,
    output dout,
    output tx_valid
  );

  modport REF (
    input clk,
    input rst_n,
    input rx_valid,
    input din,
    output dout_ref,
    output tx_valid_ref
  );


endinterface : RAM_if
```

1.18 RAM (RTL)

```
module RAM (RAM_if.DUT RAMif);

 wire[9:0] din;
 wire      clk, rst_n, rx_valid;
```

```
 reg [7:0] dout;
 reg       tx_valid;

assign RAMif.dout = dout;
assign RAMif.tx_valid = tx_valid;
assign din = RAMif.din;
assign clk = RAMif.clk;
assign rst_n = RAMif.rst_n;
assign rx_valid = RAMif.rx_valid;


reg [7:0] MEM [255:0];

reg [7:0] Rd_Addr, Wr_Addr;

always @(posedge clk) begin
    if (~rst_n) begin
        dout <= 0;
        tx_valid <= 0;
        Rd_Addr <= 0;
        Wr_Addr <= 0;
    end
    else  begin
        if (rx_valid) begin
            case (din[9:8])
                2'b00 : Wr_Addr <= din[7:0];
                2'b01 : MEM[Wr_Addr] <= din[7:0];
                2'b10 : Rd_Addr <= din[7:0];
                2'b11 : dout <= MEM[Rd_Addr];//edit
                default : dout <= 0;
            endcase
            tx_valid <= (din[9] && din[8] && rx_valid)? 1'b1 : 1'b0;//edit
        end
      end
end

endmodule
```

1.19 RAM_ref (Reference Model)

```
// Verilog module for a synchronous single-port RAM.
```

```
// This module handles read and write operations based on the SPI
commands.
module RAM_ref(RAM_if.REF RAMif);
    wire             clk;
    wire             rst_n;
    wire      [9:0]  din;
    wire             rx_valid;
     reg         tx_valid;
     reg [7:0]  dout;

assign RAMif.dout_ref = dout;
assign RAMif.tx_valid_ref = tx_valid;
assign din = RAMif.din;
assign clk = RAMif.clk;
assign rst_n = RAMif.rst_n;
assign rx_valid = RAMif.rx_valid;

   localparam  MEM_DEPTH = 256;
   localparam  ADDR_SIZE = 8;
    // --- Internal Registers and Memory Declaration ---
    // RAM memory array. We use a block RAM style for better synthesis.
    (* ram_style = "block" *)
    reg [7:0] mem[255:0];

    // Registers to hold the read and write addresses
    reg [7:0] wr_addr;
    reg [7:0] rd_addr;

    // --- Synchronous Logic (Register Updates and Memory Operations) ---
    // All operations happen on the rising edge of the clock.
    always@(posedge clk ) begin
        if (~rst_n) begin
            // Asynchronous reset
            wr_addr    <= {ADDR_SIZE{1'b0}};
            rd_addr    <= {ADDR_SIZE{1'b0}};
            tx_valid   <= 1'b0;
            dout       <= 8'd0;
        end else begin
            // Default assignments
           // tx_valid <= 1'b0;

            // Check if the SPI Slave module has a valid command
            if (rx_valid) begin
                // Decode the command from the two most significant bits
of din
```

```
                case(din[9:8])
                    // Command: '00' (Write Address)
                    2'b00: begin
                        wr_addr <= din[7:0]; // Store the new write
address
                        tx_valid <= 1'b0;
                    end
                    // Command: '01' (Write Data)
                    2'b01: begin
                        // Write the data to the previously stored address
                        mem[wr_addr] <= din[7:0];
                        tx_valid <= 1'b0;
                    end
                    // Command: '10' (Read Address)
                    2'b10: begin
                        rd_addr <= din[7:0]; // Store the new read address
                        tx_valid <= 1'b0;
                    end
                    // Command: '11' (Read Data)
                    2'b11: begin
                        tx_valid <= 1'b1;     // Assert tx_valid to signal
data is ready
                        dout <= mem[rd_addr]; // Read data from the
previously stored address
                    end
                    default: begin
                        // Do nothing for invalid commands
                    end
                endcase

            end
        end
    end

endmodule
```

2. Verification Plan

Label	Design Requirement	Description	Stimulus Generation	Functional Coverage	Functionality Check
RAM0	Reset drives all signals properly	Verify that all DUT outputs are reset to 0 when rst_n is low	Randomized rst_n (90% high, 10% low)	rst_cp bins (asserted/deasserted)	**Assertion:** reset_a

Label	Design Requirement	Description	Stimulus Generation	Functional Coverage	Functionality Check
RAM1	Opcode decoding correctness	Ensure all transaction types for din[9:8] (WRITE_ADDR, WRITE_DATA, READ_ADDR, READ_DATA) appear	Randomized din[9:8] sequence	din_cp bins val0–val3	**Scoreboard:** ram_sb.compare(golden_model)
RAM2	Transaction ordering	Check proper ordering: Write Address → Write Data → Read Address → Read Data	Randomized transaction ordering	din_trans bins (wr_d_after_wr_addr, rd_d_after_rd_addr, trans_all)	**Assertion:** _4a
RAM3	Write↔Read transitions	Ensure valid transitions WRITE_DATA→READ_ADDR and READ_DATA→WRITE_ADDR	Randomized op transitions	din_trans bins (wr_to_rd, rd_to_wr)	**Assertion:** _5a
RAM4	RX valid synchronization	Verify that RX valid signal corresponds correctly with current opcode	Randomized rx_valid toggling	cross_rx_din cross coverage	**Assertion:** _2a
RAM5	TX valid synchronization	Check TX valid high only after correct read data	Randomized tx_valid toggling	cross_tx_din bins tx_read_data	**Assertion:** _3a
RAM6	Full transaction sequence	Ensure full operation cycle (WRITE_ADDR→WRITE_DATA→READ_ADDR→READ_DATA) with TX valid	Randomized valid sequences	seq_cross bins full_seq	**Assertion:** _3a
RAM7	Address range coverage	Access low, mid, and high address ranges	Randomized addr in full memory range	addr_cp bins (low, mid, high)	**Scoreboard:** ram_sb.compare_addr()
RAM8	Data pattern coverage	Verify behavior for all-zero, all-one, and mixed data	Randomized data values	data_cp bins (zero_data, all_ones, mixed)	**Scoreboard:** ram_sb.compare_data()
RAM9	Cross opcode–address coverage	Correlate opcode with accessed address ranges	Randomized opcode and addr	cross_op_addr cross coverage	**Scoreboard:** ram_sb.cross_check_addr()
RAM10	Cross opcode–data coverage	Correlate opcode with data patterns	Randomized opcode and data	cross_op_data cross coverage	**Scoreboard:** ram_sb.cross_check_data()
RAM11	Opcode validity distribution	Check that 90% of opcodes are valid and 10% invalid	Randomized opcode generation with constraint	Coverage bins for valid/invalid	**Assertion:** _2a
RAM12	Golden model comparison	Ensure DUT output matches golden model behavior	All test cases (direct + random)	Implicit functional coverage via checks	**Scoreboard:** golden_model_check

3.Covarge

3.1 Branch Coverage

```
================================================================================
============
=== Instance: /top/dut
=== Design Unit: work.RAM
================================================================================
============
Branch Coverage:
  Enabled Coverage          Bins     Hits    Misses  Coverage
  ----------------          ----     ----    ------  --------
```

```
  Branches                  8       8       0   100.00%

================================Branch
Details================================

Branch Coverage for instance /top/dut

  Line       Item               Count     Source
  ----       ----               -----     ------
 File RAM.sv
-----------------------------------IF Branch-----------------------------------
  24                           4000     Count coming in to IF
  24          1                 123       if (~rst_n) begin

  30          1                3877       else  begin

Branch totals: 2 hits of 2 branches = 100.00%

-----------------------------------IF Branch-----------------------------------
  31                           3877     Count coming in to IF
  31          1                3440          if (rx_valid) begin

                               437     All False Count
Branch totals: 2 hits of 2 branches = 100.00%

---------------------------------CASE Branch---------------------------------
  32                           3440     Count coming in to CASE
  33          1                1013               2'b00 : Wr_Addr <= din[7:0];

  34          1                712                2'b01 : MEM[Wr_Addr] <= din[7:0];

  35          1                986                2'b10 : Rd_Addr <= din[7:0];

  36          1                729                2'b11 : dout <= MEM[Rd_Addr];//edit

Branch totals: 4 hits of 4 branches = 100.00%
```

3.2 Statement Coverage

```
Statement Coverage:
  Enabled Coverage          Bins     Hits    Misses  Coverage
  ----------------          ----     ----    ------  --------
  Statements                11       11       0   100.00%

================================Statement
Details================================

Statement Coverage for instance /top/dut --
```

```
 Line       Item               Count    Source
 ----       ----               -----    ------
File RAM.sv
 1                             module RAM (RAM_if.DUT RAMif);
 2
 3
 4
 5
 6                             wire[9:0] din;
 7                             wire     clk, rst_n, rx_valid;
 8                             reg [7:0] dout;
 9                             reg      tx_valid;
 10
 11                            assign RAMif.dout = dout;
 12                            assign RAMif.tx_valid = tx_valid;
 13                            assign din = RAMif.din;
 14         1                  8004     assign clk = RAMif.clk;
 15                            assign rst_n = RAMif.rst_n;
 16                            assign rx_valid = RAMif.rx_valid;
 17
 18
 19                            reg [7:0] MEM [255:0];
 20
 21                            reg [7:0] Rd_Addr, Wr_Addr;
 22
 23         1                  4000     always @(posedge clk) begin
```

```
  24                              if (~rst_n) begin
  25        1             123         dout <= 0;
  26        1             123         tx_valid <= 0;
  27        1             123         Rd_Addr <= 0;
  28        1             123         Wr_Addr <= 0;
  29                              end
  30                              else  begin
  31                                 if (rx_valid) begin
  32                                    case (din[9:8])
  33        1            1013             2'b00 : Wr_Addr <= din[7:0];
  34        1             712             2'b01 : MEM[Wr_Addr] <= din[7:0];
  35        1             986             2'b10 : Rd_Addr <= din[7:0];
  36        1             729             2'b11 : dout <= MEM[Rd_Addr];//edit
  37                                    default : dout <= 0;
  38                                 endcase
  39        1            3440          tx_valid <= (din[9] && din[8] && rx_valid)? 1'b1 :
1'b0;//edit
```

3.3 Toggle Coverage

```
Toggle Coverage:
    Enabled Coverage          Bins     Hits    Misses  Coverage
    ----------------          ----     ----    ------  --------
    Toggles                     76       76         0   100.00%

=================================Toggle
Details=================================

Toggle Coverage for instance /top/dut --

                              Node     1H->0L     0L->1H  "Coverage"
                              ----------------------------------
                         Rd_Addr[7-0]          1          1     100.00
                         Wr_Addr[7-0]          1          1     100.00
```

```
                clk         1         1     100.00
              din[0-9]      1         1     100.00
             dout[7-0]      1         1     100.00
               rst_n        1         1     100.00
              rx_valid      1         1     100.00
              tx_valid      1         1     100.00

Total Node Count     =       38
Toggled Node Count   =       38
Untoggled Node Count =        0

Toggle Coverage      =    100.00% (76 of 76 bins)
```

3.4 Functional Coverage

```
COVERGROUP COVERAGE:
----------------------------------------------------------------------------------------------------
Covergroup                                Metric     Goal     Bins    Status

----------------------------------------------------------------------------------------------------
 TYPE /RAM_coverage_pkg/RAM_coverage/cg          100.00%      100       -    Covered
   covered/total bins:                             14       14       -
   missing/total bins:                              0       14       -
   % Hit:                                     100.00%      100       -
   Coverpoint din_cp                          100.00%      100       -    Covered
     covered/total bins:                            4        4       -
     missing/total bins:                            0        4       -
     % Hit:                                   100.00%      100       -
   Coverpoint din_trans                       100.00%      100       -    Covered
     covered/total bins:                            3        3       -
     missing/total bins:                            0        3       -
     % Hit:                                   100.00%      100       -
   Coverpoint rx_valid_cp                     100.00%      100       -    Covered
     covered/total bins:                            1        1       -
     missing/total bins:                            0        1       -
     % Hit:                                   100.00%      100       -
   Coverpoint tx_valid_cp                     100.00%      100       -    Covered
     covered/total bins:                            1        1       -
     missing/total bins:                            0        1       -
     % Hit:                                   100.00%      100       -
   Cross cross_rx_din                         100.00%      100       -    Covered
     covered/total bins:                            4        4       -
     missing/total bins:                            0        4       -
     % Hit:                                   100.00%      100       -
   Cross cross_tx_din                         100.00%      100       -    Covered
     covered/total bins:                            1        1       -
     missing/total bins:                            0        1       -
     % Hit:                                   100.00%      100       -
 Covergroup instance \/RAM_coverage_pkg::RAM_coverage::cg
```

```
                                100.00%      100       -    Covered
    covered/total bins:                14       14       -
    missing/total bins:                 0       14       -
    % Hit:                         100.00%      100       -
    Coverpoint din_cp                      100.00%      100       -    Covered
      covered/total bins:               4        4       -
      missing/total bins:               0        4       -
      % Hit:                       100.00%      100       -
      bin val0                       1181        1       -    Covered
      bin val1                        839        1       -    Covered
      bin val2                       1140        1       -    Covered
      bin val3                        840        1       -    Covered
    Coverpoint din_trans                   100.00%      100       -    Covered
      covered/total bins:               3        3       -
      missing/total bins:               0        3       -
      % Hit:                       100.00%      100       -
      bin wr_d_after_wr_addr          598        1       -    Covered
      bin rd_d_after_rd_addr          565        1       -    Covered
      bin trans_all                    98        1       -    Covered
    Coverpoint rx_valid_cp                 100.00%      100       -    Covered
      covered/total bins:               1        1       -
      missing/total bins:               0        1       -
      % Hit:                       100.00%      100       -
      bin valid                      3547        1       -    Covered
    Coverpoint tx_valid_cp                 100.00%      100       -    Covered
      covered/total bins:               1        1       -
      missing/total bins:               0        1       -
      % Hit:                       100.00%      100       -
      bin valid                       808        1       -    Covered
    Cross cross_rx_din                     100.00%      100       -    Covered
      covered/total bins:               4        4       -
      missing/total bins:               0        4       -
      % Hit:                       100.00%      100       -
      Auto, Default and User Defined Bins:
        bin <val3,valid>              752        1       -    Covered
        bin <val2,valid>             1020        1       -    Covered
        bin <val1,valid>              732        1       -    Covered
        bin <val0,valid>             1043        1       -    Covered
    Cross cross_tx_din                     100.00%      100       -    Covered
      covered/total bins:               1        1       -
      missing/total bins:               0        1       -
      % Hit:                       100.00%      100       -
      Auto, Default and User Defined Bins:
        bin tx_read_data              749        1       -    Covered

TOTAL COVERGROUP COVERAGE: 100.00%  COVERGROUP TYPES: 1
```

3.5 Assertion Coverage

```
ASSERTION RESULTS:
---------------------------------------------------------------------
Name            File(Line)              Failure     Pass
                                        Count       Count
---------------------------------------------------------------------
/top/dut/inst_sva/reset_a
                RAM_sva.sv(5)               0       1
/top/dut/inst_sva/_2a
                RAM_sva.sv(10)              0       1
/top/dut/inst_sva/_3a
                RAM_sva.sv(13)              0       1
/top/dut/inst_sva/_4b
                RAM_sva.sv(16)              0       1
/top/dut/inst_sva/_5a
                RAM_sva.sv(18)              0       1
/RAM_wr_rd_seq_pkg/RAM_wr_rd_seq/body/#ublk#133664279#14/immed__21
                RAM_wr_rd_seq.sv(21)        0       1
/RAM_wr_only_seq_pkg/RAM_wr_only_seq/body/#ublk#227489863#14/immed__21
                RAM_wr_only_seq.sv(21)      0       1
/RAM_rd_only_seq_pkg/RAM_rd_only_seq/body/#ublk#227488423#14/immed__21
                RAM_rd_only_seq.sv(21)      0       1
```

4. Bug report

- In Read_data case write in memory read_Address not write Address

-put tx_valid assignment inside if (rx_Valid) condition

5. Waveform

Part 3: UVM Environment for SPI Wrapper

1. Code

1.1 Wrapper Top

```
import uvm_pkg::*;
`include "uvm_macros.svh"
import SPI_wrapper_test_pkg::*;
import SPI_wrapper_shared_pkg::*;


module top();

  bit clk;
  // Clock generation
  initial begin
    forever begin
      #5 clk=~clk;
    end
  end

  // Instantiate the interface and DUT
   SPI_wrapper_if SPI_wrapperif(clk);
   SPI_slave_if SPI_slaveif(clk);
   RAM_if RAMif(clk);

   WRAPPER DUT (SPI_wrapperif);
   WRAPPER_Golden REF (SPI_wrapperif);

  assign SPI_slaveif.MOSI      = SPI_wrapperif.MOSI;
  assign SPI_slaveif.rst_n     = SPI_wrapperif.rst_n;
  assign SPI_slaveif.SS_n      = SPI_wrapperif.SS_n;
  assign SPI_slaveif.tx_valid  = DUT.SLAVE_instance.tx_valid;
  assign SPI_slaveif.tx_data   = DUT.SLAVE_instance.tx_data;

    assign SPI_slaveif.rx_data      = DUT.SLAVE_instance.rx_data;
```

```
    assign SPI_slaveif.rx_valid     =
DUT.SLAVE_instance.rx_valid;
    assign SPI_slaveif.MISO         = SPI_slaveif.MISO;
    assign SPI_slaveif.rx_data_ref  = REF.SLAVE_inst.rx_data;
    assign SPI_slaveif.rx_valid_ref = REF.SLAVE_inst.rx_valid;
    assign SPI_slaveif.MISO_ref     = SPI_slaveif.MISO_ref;


    assign RAMif.din         = DUT.RAM_instance.din;
    assign RAMif.rst_n       = DUT.RAM_instance.rst_n;
    assign RAMif.rx_valid    = DUT.RAM_instance.rx_valid;
    assign RAMif.dout        = DUT.RAM_instance.dout;
    assign RAMif.tx_valid    = DUT.RAM_instance.tx_valid;
    assign RAMif.dout_ref    = REF.RAM_inst.dout;
    assign RAMif.tx_valid_ref= REF.RAM_inst.tx_valid;


  //  bind the SVA module to the design, and pass the interface
   bind WRAPPER SPI_wrapper_sva inst_sva (SPI_wrapperif);


 initial begin
   $readmemb("mem.dat", DUT.RAM_instance.MEM);
   $readmemb("mem.dat", REF.RAM_inst.mem);
  uvm_config_db #(virtual
SPI_wrapper_if)::set(null,"uvm_test_top","SPI_wrapper_if",SPI_wra
pperif);
  uvm_config_db #(virtual
SPI_slave_if)::set(null,"uvm_test_top","SPI_slave_if",SPI_slaveif
);
  uvm_config_db #(virtual
RAM_if)::set(null,"uvm_test_top","RAM_if",RAMif);
  run_test("SPI_wrapper_test");
 end

endmodule
```

1.2 Wrapper test

```
package SPI_wrapper_test_pkg;
  import uvm_pkg::*;
  `include "uvm_macros.svh"
  import SPI_wrapper_env_pkg::*;
  import SPI_wrapper_agent_pkg::*;
  import SPI_wrapper_config_pkg::*;
  import SPI_slave_env_pkg::*;
  import SPI_slave_agent_pkg::*;
  import SPI_slave_config_pkg::*;
  import RAM_env_pkg::*;
  import RAM_agent_pkg::*;
  import RAM_config_pkg::*;
  import SPI_wrapper_seq_item_pkg::*;
  import SPI_wrapper_wr_rd_seq_pkg::*;
  import SPI_wrapper_wr_only_seq_pkg::*;
  import SPI_wrapper_rd_only_seq_pkg::*;
  import SPI_wrapper_rst_seq_pkg::*;

  class SPI_wrapper_test extends uvm_test;
    `uvm_component_utils(SPI_wrapper_test)

    SPI_wrapper_env env;
    SPI_wrapper_config cfg;

    SPI_slave_env slave_env;
    SPI_slave_config slave_cfg;
    RAM_env ram_env;
    RAM_config ram_cfg;

    SPI_wrapper_wr_rd_seq wr_rd_seq;
    SPI_wrapper_wr_only_seq wr_only_seq;
    SPI_wrapper_rd_only_seq rd_only_seq;
    SPI_wrapper_rst_seq rst_seq;

    function new(string name = "SPI_wrapper_test", uvm_component parent =
null);
```

```
      super.new(name, parent);
    endfunction

    function void build_phase(uvm_phase phase);
      super.build_phase(phase);
      env = SPI_wrapper_env::type_id::create("env", this);
      cfg = SPI_wrapper_config::type_id::create("cfg");
      slave_env = SPI_slave_env::type_id::create("slave_env", this);
      slave_cfg = SPI_slave_config::type_id::create("slave_cfg");
      ram_env = RAM_env::type_id::create("ram_env", this);
      ram_cfg = RAM_config::type_id::create("ram_cfg");
      if(!uvm_config_db#(virtual SPI_wrapper_if)::get(this, "",
"SPI_wrapper_if", cfg.SPI_wrappervif))begin
        `uvm_fatal("VIF_NOT_FOUND", "You must set the VIF before building
the agent")
       end
       if(!uvm_config_db#(virtual SPI_slave_if)::get(this, "",
"SPI_slave_if", slave_cfg.SPI_slavevif))begin
        `uvm_fatal("VIF_NOT_FOUND", "You must set the VIF before building
the agent")
       end
        if(!uvm_config_db#(virtual RAM_if)::get(this, "", "RAM_if",
ram_cfg.RAMvif))begin
          `uvm_fatal("VIF_NOT_FOUND", "You must set the VIF before
building the agent")
        end
      cfg.is_active = UVM_ACTIVE;
      slave_cfg.is_active = UVM_PASSIVE;
      ram_cfg.is_active = UVM_PASSIVE;
      uvm_config_db#(SPI_wrapper_config)::set(this, "*", "CFG", cfg);
      uvm_config_db#(SPI_slave_config)::set(this, "*", "CFG", slave_cfg);
      uvm_config_db#(RAM_config)::set(this, "*", "CFG", ram_cfg);
      rst_seq  = SPI_wrapper_rst_seq::type_id::create("rst_seq");
      wr_rd_seq = SPI_wrapper_wr_rd_seq::type_id::create("wr_rd_seq");
      wr_only_seq =
SPI_wrapper_wr_only_seq::type_id::create("wr_only_seq");
      rd_only_seq =
SPI_wrapper_rd_only_seq::type_id::create("rd_only_seq");
    endfunction
```

```
    task run_phase(uvm_phase phase);
      phase.raise_objection(this);
      `uvm_info("RUN_PHASE", "Starting reset sequence", UVM_LOW)
      rst_seq.start(env.agt.sqr);

      `uvm_info("RUN_PHASE", "Starting main sequence", UVM_LOW)


      wr_only_seq.start(env.agt.sqr);
      rst_seq.start(env.agt.sqr);
      rd_only_seq.start(env.agt.sqr);
      rst_seq.start(env.agt.sqr);
      wr_rd_seq.start(env.agt.sqr);

      `uvm_info("RUN_PHASE", "Test finished", UVM_LOW)
      phase.drop_objection(this);
    endtask
  endclass : SPI_wrapper_test

endpackage : SPI_wrapper_test_pkg
```

1.3 Wrapper env

```
package SPI_wrapper_env_pkg;
  import uvm_pkg::*;
  `include "uvm_macros.svh"
  import SPI_wrapper_agent_pkg::*;
  import SPI_wrapper_scoreboard_pkg::*;
  import SPI_wrapper_coverage_pkg::*;

  class SPI_wrapper_env extends uvm_env;
    `uvm_component_utils(SPI_wrapper_env)
    SPI_wrapper_agent agt;
    SPI_wrapper_scoreboard sb;
    SPI_wrapper_coverage cov;

    function new(string name = "SPI_wrapper_env", uvm_component
parent = null);
```

```
    super.new(name, parent);
  endfunction

  function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    agt = SPI_wrapper_agent::type_id::create("agt", this);
    sb  = SPI_wrapper_scoreboard::type_id::create("sb", this);
    cov = SPI_wrapper_coverage::type_id::create("cov", this);
  endfunction

  function void connect_phase(uvm_phase phase);
    super.connect_phase(phase);
    agt.agt_ap.connect(sb.sb_export);
    agt.agt_ap.connect(cov.cov_export);
  endfunction
endclass : SPI_wrapper_env

endpackage : SPI_wrapper_env_pkg
```

1.4 Wrapper agent

```
package SPI_wrapper_agent_pkg;
  import uvm_pkg::*;
  `include "uvm_macros.svh"
  import SPI_wrapper_config_pkg::*;
  import SPI_wrapper_driver_pkg::*;
  import SPI_wrapper_monitor_pkg::*;
  import SPI_wrapper_sequencer_pkg::*;
  import SPI_wrapper_seq_item_pkg::*;

  class SPI_wrapper_agent extends uvm_agent;
    `uvm_component_utils(SPI_wrapper_agent)

    SPI_wrapper_config cfg;
    SPI_wrapper_driver drv;
    SPI_wrapper_sequencer sqr;
    SPI_wrapper_monitor mon;
    uvm_analysis_port#(SPI_wrapper_seq_item) agt_ap;
```

```
    function new(string name = "SPI_wrapper_agent", uvm_component
parent = null);
      super.new(name, parent);
    endfunction

    function void build_phase(uvm_phase phase);
      super.build_phase(phase);
      if (!uvm_config_db#(SPI_wrapper_config)::get(this, "",
"CFG", cfg))
        `uvm_fatal("CFG_NOT_FOUND", "You must set the CFG before
building the agent")

      if (cfg.is_active == UVM_ACTIVE) begin
        drv = SPI_wrapper_driver::type_id::create("drv", this);
        sqr = SPI_wrapper_sequencer::type_id::create("sqr",
this);
      end
      mon = SPI_wrapper_monitor::type_id::create("mon", this);
      agt_ap = new("agt_ap", this);
    endfunction

    function void connect_phase(uvm_phase phase);
      super.connect_phase(phase);
      if (cfg.is_active == UVM_ACTIVE) begin
        drv.seq_item_port.connect(sqr.seq_item_export);
        // TODO: set driver's virtual interface
        drv.vif = cfg.SPI_wrappervif;
      end
      // TODO: connect monitor vif
      mon.vif = cfg.SPI_wrappervif;
      mon.mon_ap.connect(agt_ap);
    endfunction
  endclass : SPI_wrapper_agent

endpackage : SPI_wrapper_agent_pkg
```

1.5 Wrapper driver

```
package SPI_wrapper_driver_pkg;
```

```
  import uvm_pkg::*;
  `include "uvm_macros.svh"
  import SPI_wrapper_seq_item_pkg::*;
  import SPI_wrapper_config_pkg::*;

  class SPI_wrapper_driver extends
uvm_driver#(SPI_wrapper_seq_item);
    `uvm_component_utils(SPI_wrapper_driver)
    virtual SPI_wrapper_if vif;
    SPI_wrapper_seq_item tx;

    function new(string name = "SPI_wrapper_driver",
uvm_component parent = null);
      super.new(name, parent);
    endfunction

    task run_phase(uvm_phase phase);

      forever begin
        seq_item_port.get_next_item(tx);
        // TODO: Drive DUT signals using tx fields
        vif.rst_n      = tx.rst_n;
        vif.SS_n       = tx.SS_n;
        vif.MOSI       = tx.MOSI;
        @(negedge vif.clk);
        seq_item_port.item_done();
      end
    endtask
  endclass : SPI_wrapper_driver

endpackage : SPI_wrapper_driver_pkg
```

1.6 Wrapper monitor

```
package SPI_wrapper_monitor_pkg;
  import uvm_pkg::*;
  `include "uvm_macros.svh"
  import SPI_wrapper_seq_item_pkg::*;
```

```
  class SPI_wrapper_monitor extends uvm_monitor;
    `uvm_component_utils(SPI_wrapper_monitor)

    virtual SPI_wrapper_if vif;
    SPI_wrapper_seq_item tx;
    uvm_analysis_port#(SPI_wrapper_seq_item) mon_ap;

    function new(string name = "SPI_wrapper_monitor",
uvm_component parent = null);
      super.new(name, parent);
    endfunction
function void build_phase(uvm_phase phase);
     super.build_phase(phase);
        mon_ap=new("mon_ap",this);
    endfunction
    task run_phase(uvm_phase phase);

      forever begin
        tx = SPI_wrapper_seq_item::type_id::create("tx");
        @(negedge vif.clk);
        // TODO: sample DUT outputs and fill tx fields
        tx.MOSI       = vif.MOSI;
        tx.SS_n      = vif.SS_n;
        tx.rst_n    = vif.rst_n;
        tx.MISO      = vif.MISO;
        tx.MISO_ref  = vif.MISO_ref;
        `uvm_info("MONITOR", $sformatf("Observed: %s",
tx.convert2string()), UVM_LOW)
        mon_ap.write(tx);
      end
    endtask
  endclass : SPI_wrapper_monitor

endpackage : SPI_wrapper_monitor_pkg
```

1.7 Wrapper sequencer

```
package SPI_wrapper_sequencer_pkg;
  import uvm_pkg::*;
```

```
  `include "uvm_macros.svh"
  import SPI_wrapper_seq_item_pkg::*;

  class SPI_wrapper_sequencer extends
uvm_sequencer#(SPI_wrapper_seq_item);
    `uvm_component_utils(SPI_wrapper_sequencer)
    function new(string name = "SPI_wrapper_sequencer",
uvm_component parent = null);
      super.new(name, parent);
    endfunction
  endclass

endpackage : SPI_wrapper_sequencer_pkg
```

1.8 Wrapper scoreboard

```
package SPI_wrapper_scoreboard_pkg;
  import uvm_pkg::*;
  `include "uvm_macros.svh"
  import SPI_wrapper_seq_item_pkg::*;

  class SPI_wrapper_scoreboard extends uvm_scoreboard;
    `uvm_component_utils(SPI_wrapper_scoreboard)
    uvm_analysis_port#(SPI_wrapper_seq_item) sb_export;
    uvm_tlm_analysis_fifo#(SPI_wrapper_seq_item) sb_fifo;
    SPI_wrapper_seq_item received;

    function new(string name = "SPI_wrapper_scoreboard",
uvm_component parent = null);
      super.new(name, parent);
    endfunction

    function void build_phase(uvm_phase phase);
      super.build_phase(phase);
      sb_export = new("sb_export", this);
      sb_fifo   = new("sb_fifo", this);
    endfunction
     function void connect_phase(uvm_phase phase);
      super.connect_phase(phase);
```

```
      sb_export.connect(sb_fifo.analysis_export);
    endfunction

    task run_phase(uvm_phase phase);
      forever begin
        sb_fifo.get(received);
        golden_model(received);
      end
    endtask

    // TODO: Implement golden_model tailored to your DUT's
expected behavior
    task automatic golden_model(input SPI_wrapper_seq_item tx);
      // Example: simple pass-through model (replace with actual
model logic)
      if (tx.MISO !== tx.MISO_ref) begin
        `uvm_error("SCOREBOARD", $sformatf("Mismatch detected!
MISO=%0d, Expected MISO_ref=%0d", tx.MISO, tx.MISO_ref))
      end else begin
        `uvm_info("SCOREBOARD", "Match confirmed between MISO and
MISO_ref", UVM_LOW)
      end
     // `uvm_info("SCOREBOARD", $sformatf("Golden model called
for: %s", tx.convert2string()), UVM_LOW)
    endtask
  endclass : SPI_wrapper_scoreboard

endpackage : SPI_wrapper_scoreboard_pkg
```

1.9 Wrapper coverage

```
package SPI_wrapper_coverage_pkg;
  import uvm_pkg::*;
  `include "uvm_macros.svh"
  import SPI_wrapper_seq_item_pkg::*;

  class SPI_wrapper_coverage extends uvm_component;
    `uvm_component_utils(SPI_wrapper_coverage)
    uvm_analysis_port#(SPI_wrapper_seq_item) cov_export;
```

```
    uvm_tlm_analysis_fifo#(SPI_wrapper_seq_item) cov_fifo;
    SPI_wrapper_seq_item cov_item;

    covergroup cg;

coverpoint cov_item.MOSI {
       bins low = {0};
       bins high = {1};
     }

     coverpoint cov_item.SS_n {
       bins inactive = {1};
       bins active = {0};
     }

     coverpoint cov_item.rst_n {
       bins deasserted = {1};
       bins asserted = {0};
     }

     coverpoint cov_item.MISO {
       bins low = {0};
       bins high = {1};
     }




    endgroup : cg

    function new(string name = "SPI_wrapper_coverage",
uvm_component parent = null);
      super.new(name, parent);
      cg = new();
    endfunction

    function void build_phase(uvm_phase phase);
      super.build_phase(phase);
      cov_export = new("cov_export", this);
```

```
      cov_fifo   = new("cov_fifo", this);
    endfunction
    function void connect_phase(uvm_phase phase);
      super.connect_phase(phase);
      cov_export.connect(cov_fifo.analysis_export);
    endfunction

    task run_phase(uvm_phase phase);
      forever begin
        cov_fifo.get(cov_item);
        cg.sample();
      end
    endtask
  endclass : SPI_wrapper_coverage

endpackage : SPI_wrapper_coverage_pkg
```

1.10 Wrapper config

```
package SPI_wrapper_config_pkg;

  import uvm_pkg::*;
  `include "uvm_macros.svh"

  class SPI_wrapper_config extends uvm_object;
    `uvm_object_utils(SPI_wrapper_config)

    // TODO: Add your virtual interface handle(s) here
    // Example:
     virtual SPI_wrapper_if SPI_wrappervif;
    // TODO: Add other config fields (e.g. is_active,
clock_period, time_scale...)
    uvm_active_passive_enum is_active;

    function new (string name = "SPI_wrapper_config");
      super.new(name);
    endfunction
  endclass
```

```
endpackage : SPI_wrapper_config_pkg
```

1.11 Wrapper seq item

```
package SPI_wrapper_seq_item_pkg;
  import uvm_pkg::*;
  `include "uvm_macros.svh"
  import SPI_wrapper_shared_pkg::*;

  class SPI_wrapper_seq_item extends uvm_sequence_item;
    `uvm_object_utils(SPI_wrapper_seq_item)

    // TODO: add randomized stimulus fields that match your DUT
   rand bit MOSI, SS_n, rst_n;
   bit MISO;
   bit MISO_ref;


  function new(string name = "SPI_wrapper_seq_item");
      super.new(name);
    endfunction

    function string convert2string();
      return $sformatf("%s MOSI=%0d SS_n=%0d rst_n=%0d MISO=%0d
MISO_ref=%0d",
        super.convert2string(), MOSI, SS_n, rst_n, MISO,
MISO_ref);
    endfunction

    // TODO: Add constraints tailored to DUT behavior
    constraint rst_n_mostly_deasserted{ rst_n dist{1:=99,0:=1};}

    rand bit [10:0] MO_data; // 11-bit array for MOSI data
    static bit [10:0] temp_MO_data;
    static bit [10:0] prev_MO_data;
    static int cycle_count = 0;
    constraint valid_comb1 {
      MO_data[10:8] inside {3'b000, 3'b001, 3'b110, 3'b111};
```

```
    }
      constraint valid_comb2{
                    MOSI inside {0};
                    SS_n inside {1};
                  }

     constraint write_only_seq { MO_data[10:8] inside {3'b000,
3'b001}; }
     constraint read_only_seq  {
      MO_data[10:8] inside {3'b110, 3'b111};
      (prev_MO_data[8]==0)-> MO_data[8]==1;
      (prev_MO_data[8]==1)-> MO_data[8]==0;
       }

        bit[1:0] wr_op[] = '{WRITE_ADDR, WRITE_DATA};
        bit[1:0] rd_op[] = '{READ_ADDR, READ_DATA};

        constraint rw_rd_seq {
         (prev_MO_data[9:8] == WRITE_ADDR) -> MO_data[9:8] inside
{wr_op};
         (prev_MO_data[9:8] == READ_ADDR)  -> MO_data[9:8] inside
{rd_op};
         (prev_MO_data[9:8] == WRITE_DATA) -> MO_data[9:8]  dist
{READ_ADDR:=60 , WRITE_ADDR:=40};
         (prev_MO_data[9:8] == READ_DATA)  -> MO_data[9:8]  dist
{WRITE_ADDR:=60 , READ_ADDR:=40};
          }



    function void post_randomize();
     //MO_data[7:0] = 8'h54;
    if (cycle_count>=1 && cycle_count<=23) begin SS_n = 0; end
    if (cycle_count>=2 && cycle_count<=12) begin MOSI =
temp_MO_data[12-cycle_count]; end
      prev_MO_data = temp_MO_data;
     `uvm_info("POST_RANDOMIZE", $sformatf("Generated MO_data:
0x%0h, cycle_count=%0d", temp_MO_data,cycle_count), UVM_LOW)
```

```
      `uvm_info("POST_RANDOMIZE", $sformatf("MOSI=%0d SS_n=%0d
rst_n=%0d", MOSI, SS_n, rst_n), UVM_LOW)
     if (cycle_count == 0) begin
        temp_MO_data = MO_data;
        SS_n = 1; MOSI = 0;
        cycle_count++;
      end
      else if (cycle_count==13&& temp_MO_data[9:8]!=2'b11) begin
         cycle_count = 0;
      end
      else if (cycle_count==23&& temp_MO_data[9:8]==2'b11) begin
         cycle_count = 0;
      end
      else begin
         cycle_count++;
      end

      if (!rst_n)
      begin
        MOSI = 0;
        SS_n = 1;
        cycle_count=0;
        prev_MO_data[10:8]=3'b111;
        temp_MO_data[10:8]=3'b111;
      end


    endfunction


  endclass : SPI_wrapper_seq_item

endpackage : SPI_wrapper_seq_item_pkg
```

1.12 Wrapper rst seq

```
package SPI_wrapper_rst_seq_pkg;
```

```
  import uvm_pkg::*;
  `include "uvm_macros.svh"
  import SPI_wrapper_seq_item_pkg::*;

  class SPI_wrapper_rst_seq extends uvm_sequence
#(SPI_wrapper_seq_item);
    `uvm_object_utils(SPI_wrapper_rst_seq)

    function new(string name = "SPI_wrapper_rst_seq");
      super.new(name);
    endfunction

    task body;
      SPI_wrapper_seq_item tx =
SPI_wrapper_seq_item::type_id::create("tx");
      start_item(tx);
      tx.rst_n    = 0; // Assert reset
      finish_item(tx);
      // TODO: allow DUT to settle after reset
    endtask

  endclass : SPI_wrapper_rst_seq

endpackage : SPI_wrapper_rst_seq_pkg
```

1.13 Wrapper write only seq

```
package SPI_wrapper_wr_only_seq_pkg;
  import uvm_pkg::*;
  `include "uvm_macros.svh"
  import SPI_wrapper_seq_item_pkg::*;

  class SPI_wrapper_wr_only_seq extends uvm_sequence #(SPI_wrapper_seq_item);
    `uvm_object_utils(SPI_wrapper_wr_only_seq)
     SPI_wrapper_seq_item tx;
    function new(string name = "SPI_wrapper_wr_only_seq");
      super.new(name);
```

```
    endfunction

    task body;

      repeat (500) begin
        tx = SPI_wrapper_seq_item::type_id::create("tx");
        start_item(tx);
        tx.constraint_mode(1);
        tx.write_only_seq.constraint_mode(1);
        tx.rw_rd_seq.constraint_mode(0);
        tx.read_only_seq.constraint_mode(0);
        assert(tx.randomize()) else `uvm_fatal("RANDOMIZE_FAIL","randomization
failed");
        finish_item(tx);
      end
    endtask
  endclass : SPI_wrapper_wr_only_seq

endpackage : SPI_wrapper_wr_only_seq_pkg
```

1.14 Wrapper read only seq

```
package SPI_wrapper_rd_only_seq_pkg;
  import uvm_pkg::*;
  `include "uvm_macros.svh"
  import SPI_wrapper_seq_item_pkg::*;

  class SPI_wrapper_rd_only_seq extends uvm_sequence
#(SPI_wrapper_seq_item);
    `uvm_object_utils(SPI_wrapper_rd_only_seq)
     SPI_wrapper_seq_item tx;
    function new(string name = "SPI_wrapper_rd_only_seq");
      super.new(name);
    endfunction

    task body;

      repeat (500) begin
```

```
        tx = SPI_wrapper_seq_item::type_id::create("tx");
        start_item(tx);
        tx.constraint_mode(1);
        tx.write_only_seq.constraint_mode(0);
        tx.rw_rd_seq.constraint_mode(0);
        tx.read_only_seq.constraint_mode(1);
        assert(tx.randomize()) else
`uvm_fatal("RANDOMIZE_FAIL","randomization failed");
        finish_item(tx);
      end
    endtask
  endclass : SPI_wrapper_rd_only_seq

endpackage : SPI_wrapper_rd_only_seq
```

1.15 Wrapper write read seq

```
package SPI_wrapper_wr_rd_seq_pkg;
  import uvm_pkg::*;
  `include "uvm_macros.svh"
  import SPI_wrapper_seq_item_pkg::*;

  class SPI_wrapper_wr_rd_seq extends uvm_sequence
#(SPI_wrapper_seq_item);
    `uvm_object_utils(SPI_wrapper_wr_rd_seq)
     SPI_wrapper_seq_item tx;
    function new(string name = "SPI_wrapper_wr_rd_seq");
      super.new(name);
    endfunction

    task body;

      repeat (500) begin
        tx = SPI_wrapper_seq_item::type_id::create("tx");
        start_item(tx);
        tx.constraint_mode(1);
        tx.write_only_seq.constraint_mode(0);
        tx.rw_rd_seq.constraint_mode(1);
        tx.read_only_seq.constraint_mode(0);
```

```
        assert(tx.randomize()) else
`uvm_fatal("RANDOMIZE_FAIL","randomization failed");
        finish_item(tx);
      end
    endtask
  endclass : SPI_wrapper_wr_rd_seq

endpackage : SPI_wrapper_wr_rd_seq_pkg
```

1.16 Wrapper SVA

```
import SPI_wrapper_shared_pkg::*;
module SPI_wrapper_sva(SPI_wrapper_if.DUT SPI_wrapperif);




  MISO_stable_a: assert property (@(posedge SPI_wrapperif.clk)
disable iff (!SPI_wrapperif.rst_n)
    $fell
(SPI_wrapperif.SS_n)|=>$stable(SPI_wrapperif.MISO)[*13]);

endmodule
```

1.17 Wrapper if

```
interface SPI_wrapper_if (input logic clk);
  // TODO: Replace/add signals to match your DUT interface.

bit MOSI, SS_n, rst_n;
bit MISO;
bit MISO_ref;

  modport DUT (
    input clk,
    input rst_n,
    input SS_n,
```

```
    input MOSI,
    output MISO
  );
  modport REF (
    input clk,
    input rst_n,
    input SS_n,
    input MOSI,
    output MISO_ref
  );

endinterface : SPI_wrapper_if
```

1.18 Wrapper (RTL)

```
module WRAPPER (SPI_wrapper_if.DUT SPI_wrapperif);

wire MOSI, SS_n, clk, rst_n;
wire MISO;

assign SPI_wrapperif.MISO = MISO;

assign MOSI = SPI_wrapperif.MOSI;
assign SS_n = SPI_wrapperif.SS_n;
assign clk = SPI_wrapperif.clk;
assign rst_n = SPI_wrapperif.rst_n;

wire [9:0] rx_data_din;
wire       rx_valid;
wire       tx_valid;
wire [7:0] tx_data_dout;

RAM_   RAM_instance   (rx_data_din,clk,rst_n,rx_valid,tx_data_dou
t,tx_valid);
SLAVE SLAVE_instance
(MOSI,MISO,SS_n,clk,rst_n,rx_data_din,rx_valid,tx_data_dout,tx_va
lid);
`ifdef SIM
```

```
reset_a :assert property (@(posedge SPI_wrapperif.clk)
!SPI_wrapperif.rst_n |->
  ##1 !SPI_wrapperif.MISO && !rx_valid &&rx_valid==0);

`endif  // SIM
endmodule
```

1.19 Wrapper ref (ref model)

```
module WRAPPER_Golden (SPI_wrapper_if.REF SPI_wrapperif);

wire MOSI, SS_n, clk, rst_n;
wire MISO;

assign SPI_wrapperif.MISO_ref = MISO;

assign MOSI = SPI_wrapperif.MOSI;
assign SS_n = SPI_wrapperif.SS_n;
assign clk = SPI_wrapperif.clk;
assign rst_n = SPI_wrapperif.rst_n;

wire [9:0] rx_data_din;
wire       rx_valid;
wire       tx_valid;
wire [7:0] tx_data_dout;

RAM_ref   RAM_inst   (rx_data_din,clk,rst_n,rx_valid,tx_data_dout
,tx_valid);
SPI_Slave_Golden
SLAVE_inst(MOSI,MISO,SS_n,clk,rst_n,rx_data_din,rx_valid,tx_data_
dout,tx_valid);

endmodule
```

2. Verification Plan

Label	Design Requirement	Description	Stimulus Generation	Functional Coverage	Functionality Check
SPI0	WR-REQ-001: Write transactions	DUT accepts write address/data when SS_n active	Randomized write items: SPI_wrapper_wr_only_seq.sv; SPI_wrapper_wr_rd_seq.sv (fields in SPI_wrapper_seq_item.sv)	SPI_wrapper_coverage.sv coverpoints (MOSI, SS_n); add MO_data[9:8] opcode bins	Scoreboard SPI_wrapper_scoreboard.sv (extend golden_model to check RAM contents)
SPI1	RD-REQ-001: Read transactions	DUT drives read data on MISO and rx_valid on reads	Randomized read items: SPI_wrapper_rd_only_seq.sv; SPI_wrapper_wr_rd_seq.sv (SPI_wrapper_seq_item.sv)	SPI_wrapper_coverage.sv coverpoints (MISO, SS_n); add opcode & data-value bins	Check MISO vs reference (MISO_ref/RAMif.dout_ref) in SPI_wrapper_scoreboard.sv
SPI2	RST-REQ-001: Reset behavior	All outputs quiescent and internal state reset when rst_n asserted	SPI_wrapper_rst_seq.sv and seq_item rst_n constraints (SPI_wrapper_seq_item.sv)	SPI_wrapper_coverage.sv rst_n bins (asserted/deasserted); cross with SS_n	SVA reset_a (in SPI_wrapper_sva.sv.bak) and monitor checks; scoreboard should observe no transactions during reset
SPI3	SVA-REQ-001: MISO stability	MISO must remain stable for N cycles after SS_n edge	Triggered by normal sequences (wr/rd) observed by SPI_wrapper_monitor.sv	coverage: add cross SS_n edge ¦— MISO change bin	Assertion MISO_stable_a in SPI_wrapper_sva.sv (bound via top.sv bind)
SPI4	DRV-REQ-001: Driver timing	Driver applies item fields to DUT at correct clock edges	Driver SPI_wrapper_driver.sv reads items and drives vif.rst_n, vif.SS_n, vif.MOSI	coverage: driver timing bins / action-on-edge bins (add)	Monitor SPI_wrapper_monitor.sv samples signals; add timing checks comparing expected edge vs actual

3. Coverage

3.1 Branch coverage

Branch Coverage:

Enabled Coverage	Bins	Hits	Misses	Coverage
Branches	38	38	0	100.00%

==========Branch Details==========

Branch Coverage for instance /top/DUT/SLAVE_instance

Line	Item	Count	Source
File SPI_slave.sv			
----------IF Branch----------			
20		415	Count coming in to IF
20	1	18	if (~rst_n) begin
23	1	397	else begin

Branch totals: 2 hits of 2 branches = 100.00%

----------CASE Branch----------			
30		1077	Count coming in to CASE
31	1	202	IDLE : begin
37	1	146	CHK_CMD : begin
51	1	374	WRITE : begin
57	1	188	READ_ADD : begin
63	1	166	READ_DATA : begin
69	1	1	default : ;

Branch totals: 6 hits of 6 branches = 100.00%

----------IF Branch----------			
32		202	Count coming in to IF

32 1 101 if (SS_n)

34 1 101 else

Branch totals: 2 hits of 2 branches = 100.00%

----------IF Branch----------

38 146 Count coming in to IF

40 1 146 else begin

Branch totals: 1 hit of 1 branch = 100.00%

----------IF Branch----------

41 146 Count coming in to IF

41 1 99 if (~MOSI)

43 1 47 else begin

Branch totals: 2 hits of 2 branches = 100.00%

----------IF Branch----------

44 47 Count coming in to IF

44 1 25 if (!received_address) //edit

46 1 22 else

Branch totals: 2 hits of 2 branches = 100.00%

----------IF Branch----------

```
  52                    374    Count coming in to IF
  52          1          51          if (SS_n)
  54          1         323           else

Branch totals: 2 hits of 2 branches = 100.00%

-----------------------------------IF Branch-----------------------------------
  58                    188    Count coming in to IF
  58          1          24          if (SS_n)
  60          1         164           else

Branch totals: 2 hits of 2 branches = 100.00%

-----------------------------------IF Branch-----------------------------------
  64                    166    Count coming in to IF
  64          1          22          if (SS_n)
  66          1         144           else

Branch totals: 2 hits of 2 branches = 100.00%

-----------------------------------IF Branch-----------------------------------
  74                   1488    Count coming in to IF
  74          1          18      if (~rst_n) begin
  80          1        1470       else begin

Branch totals: 2 hits of 2 branches = 100.00%
```

----------CASE Branch----------

81		1470	Count coming in to CASE
82	1	100	IDLE : begin
85	1	99	CHK_CMD : begin
88	1	597	WRITE : begin
97	1	263	READ_ADD : begin
107	1	411	READ_DATA : begin

Branch totals: 5 hits of 5 branches = 100.00%

----------IF Branch----------

89		597	Count coming in to IF
89	1	506	if (counter > 0) begin
93	1	91	else begin

Branch totals: 2 hits of 2 branches = 100.00%

----------IF Branch----------

98		263	Count coming in to IF
98	1	221	if (counter > 0) begin
102	1	42	else begin

Branch totals: 2 hits of 2 branches = 100.00%

----------IF Branch----------

108		411	Count coming in to IF
108	1	163	if (tx_valid) begin
118	1	248	else begin

Branch totals: 2 hits of 2 branches = 100.00%

----------IF Branch----------

110		163	Count coming in to IF
110	1	131	if (counter > 0) begin
114	1	32	else begin

Branch totals: 2 hits of 2 branches = 100.00%

----------IF Branch----------

119		248	Count coming in to IF
119	1	228	if (counter > 0) begin
124	1	20	else begin

Branch totals: 2 hits of 2 branches = 100.00%

Condition Coverage:

```
  Enabled Coverage          Bins   Covered    Misses  Coverage
  ----------------          ----     ----   ------ --------
  Conditions                   4       4       0   100.00%


================================================================================
===========
=== Instance: /top/DUT/RAM_instance
=== Design Unit: work.RAM_
================================================================================
===========
Branch Coverage:
  Enabled Coverage          Bins     Hits    Misses  Coverage
  ----------------          ----     ----   ------ --------
  Branches                     8       8       0   100.00%


================================Branch
Details================================


Branch Coverage for instance /top/DUT/RAM_instance


  Line       Item               Count     Source
  ----       ----               -----     ------
 File RAM.sv
------------------------------------IF Branch-------------------------------------
  14                            711     Count coming in to IF
  14          1                  18       if (~rst_n) begin
  20          1                 693       else  begin
```

Branch totals: 2 hits of 2 branches = 100.00%

----------IF Branch----------

21		693	Count coming in to IF
21	1	149	if (rx_valid) begin
		544	All False Count

Branch totals: 2 hits of 2 branches = 100.00%

----------CASE Branch----------

22		149	Count coming in to CASE
23	1	42	2'b00 : Wr_Addr <= din[7:0];
24	1	47	2'b01 : MEM[Wr_Addr] <= din[7:0];
25	1	43	2'b10 : Rd_Addr <= din[7:0];
26	1	17	2'b11 : dout <= MEM[Rd_Addr];//edit

Branch totals: 4 hits of 4 branches = 100.00%

3.2 Statement coverage

==========Statement Details==========

Statement Coverage for instance /top/DUT/RAM_instance --

Line	Item	Count	Source

File RAM.sv

```
1                                   module RAM_ (din,clk,rst_n,rx_valid,dout,tx_valid);
2
3                                   input     [9:0] din;
4                                   input         clk, rst_n, rx_valid;
5
6                                   output reg [7:0] dout;
7                                   output reg      tx_valid;
8
9                                   reg [7:0] MEM [255:0];
10
11                                   reg [7:0] Rd_Addr, Wr_Addr;
12
13          1               711     always @(posedge clk) begin
14                                      if (~rst_n) begin
15          1               18          dout <= 0;
16          1               18          tx_valid <= 0;
17          1               18          Rd_Addr <= 0;
18          1               18          Wr_Addr <= 0;
19                                      end
20                                      else  begin
21                                        if (rx_valid) begin
22                                          case (din[9:8])
23          1               42                2'b00 : Wr_Addr <= din[7:0];
24          1               47                2'b01 : MEM[Wr_Addr] <= din[7:0];
25          1               43                2'b10 : Rd_Addr <= din[7:0];
26          1               17                2'b11 : dout <= MEM[Rd_Addr];//edit
```

```
  27                                      default : dout <= 0;
  28                                    endcase
  29          1                149            tx_valid <= (din[9] && din[8] && rx_valid)? 1'b1 :
1'b0;//edit
Statement Coverage:
  Enabled Coverage          Bins     Hits    Misses  Coverage
  ----------------          ----     ----    ------  --------
  Statements                38       38       0   100.00%
Statement Coverage for instance /top/DUT/SLAVE_instance --
  Line       Item                Count     Source
  ----       ----                -----     ------
 File SPI_slave.sv
  1                                 module SLAVE
(MOSI,MISO,SS_n,clk,rst_n,rx_data,rx_valid,tx_data,tx_valid);
  2
  3                                 localparam IDLE     = 3'b000;
  4                                 localparam CHK_CMD   = 3'b001;
  5                                 localparam WRITE    = 3'b010;//edit
  6                                 localparam READ_ADD  = 3'b011;
  7                                 localparam READ_DATA = 3'b100;
  8
  9                                 input         MOSI, clk, rst_n, SS_n, tx_valid;
  10                                 input     [7:0] tx_data;
  11                                 output reg [9:0] rx_data;
  12                                 output reg      rx_valid, MISO;
  13
```

```
14                          reg [3:0] counter;
15                          reg      received_address;
16
17                          reg [2:0] cs, ns;
18
19      1           415     always @(posedge clk) begin
20                              if (~rst_n) begin
21      1            18             cs <= IDLE;
22                              end
23                              else begin
24      1           397             cs <= ns;
25                              end
26                          end
27
28      1          1077     always @(*) begin
29
30                              case (cs)
31                                  IDLE : begin
32                                      if (SS_n)
33      1           101                     ns = IDLE;
34                                      else
35      1           101                     ns = CHK_CMD;
36                                  end
37                                  CHK_CMD : begin
38                                      if (SS_n)
39                                          ns = IDLE;
```

```
40                                  else begin
41                                    if (~MOSI)
42      1           99                    ns = WRITE;
43                                    else begin
44                                      if (!received_address) //edit
45      1           25                      ns = READ_ADD;
46                                      else
47      1           22                      ns = READ_DATA;
48                                    end
49                                  end
50                                end
51                                WRITE : begin
52                                  if (SS_n)
53      1           51                ns = IDLE;
54                                  else
55      1           323               ns = WRITE;
56                                end
57                                READ_ADD : begin
58                                  if (SS_n)
59      1           24                ns = IDLE;
60                                  else
61      1           164               ns = READ_ADD;
62                                end
63                                READ_DATA : begin
64                                  if (SS_n)
65      1           22                ns = IDLE;
```

```
66                                  else
67      1           144                 ns = READ_DATA;
68                                  end
69                                  default : ;
70                              endcase
71                          end
72
73      1           1488    always @(posedge clk) begin
74                              if (~rst_n) begin
75      1            18             rx_data <= 0;
76      1            18             rx_valid <= 0;
77      1            18             received_address <= 0;
78      1            18             MISO <= 0;
79                              end
80                              else begin
81                                  case (cs)
82                                      IDLE : begin
83      1           100                 rx_valid <= 0;
84                                      end
85                                      CHK_CMD : begin
86      1            99                 counter <= 10;
87                                      end
88                                      WRITE : begin
89                                          if (counter > 0) begin
90      1           506                     rx_data[counter-1] <= MOSI;
91      1           506                     counter <= counter - 1;
```

```
92                                  end
93                                  else begin
94          1           91                  rx_valid <= 1;
95                                  end
96                                end
97                                READ_ADD : begin
98                                  if (counter > 0) begin
99          1           221                 rx_data[counter-1] <= MOSI;
100         1           221                 counter <= counter - 1;
101                                 end
102                                 else begin
103         1           42                  rx_valid <= 1;
104         1           42                  received_address <= 1;
105                               end
106                               end
107                               READ_DATA : begin
108                                 if (tx_valid) begin
109         1           163                 rx_valid <= 0;
110                                   if (counter > 0) begin
111         1           131                     MISO <= tx_data[counter-1];
112         1           131                     counter <= counter - 1;
113                                   end
114                                   else begin
115         1           32                      received_address <= 0;
116                                   end
117                                 end
```

118				else begin
119				if (counter > 0) begin
120	1		228	rx_data[counter-1] <= MOSI;
121	1		228	counter <= counter - 1;
122	1		228	rx_valid <= 0;///
123				end
124				else begin
125	1		20	rx_valid <= 1;
126	1		20	counter <= 9;//edit

3.3 Toggle coverage

Toggle Coverage:

Enabled Coverage	Bins	Hits	Misses	Coverage
Toggles	50	50	0	100.00%

==========Toggle Details==========

Toggle Coverage for instance /top/DUT --

Node	1H->0L	0L->1H	"Coverage"
MISO	1	1	100.00
MOSI	1	1	100.00
SS_n	1	1	100.00
clk	1	1	100.00
rst_n	1	1	100.00

rx_data_din[0-9] 1 1 100.00

rx_valid 1 1 100.00

tx_data_dout[0-7] 1 1 100.00

tx_valid 1 1 100.00

Total Node Count = 25

Toggled Node Count = 25

Untoggled Node Count = 0

Toggle Coverage = 100.00% (50 of 50 bins)

3.4 Functional coverage

==========

=== Instance: /SPI_wrapper_coverage_pkg

=== Design Unit: work.SPI_wrapper_coverage_pkg

==========

Covergroup Coverage:

Covergroups 1 na na 100.00%

Coverpoints/Crosses 4 na na na

Covergroup Bins 8 8 0 100.00%

Covergroup Metric Goal Bins Status

TYPE /SPI_wrapper_coverage_pkg/SPI_wrapper_coverage/cg

100.00% 100 - Covered

covered/total bins:	8	8	-	
missing/total bins:	0	8	-	
% Hit:	100.00%	100	-	
Coverpoint #coverpoint_0#	100.00%	100	-	Covered
covered/total bins:	2	2	-	
missing/total bins:	0	2	-	
% Hit:	100.00%	100	-	
bin low	1012	1	-	Covered
bin high	491	1	-	Covered
Coverpoint #coverpoint_1#	100.00%	100	-	Covered
covered/total bins:	2	2	-	
missing/total bins:	0	2	-	
% Hit:	100.00%	100	-	
bin inactive	117	1	-	Covered
bin active	1386	1	-	Covered
Coverpoint #coverpoint_2#	100.00%	100	-	Covered
covered/total bins:	2	2	-	
missing/total bins:	0	2	-	
% Hit:	100.00%	100	-	
bin deasserted	1485	1	-	Covered
bin asserted	18	1	-	Covered
Coverpoint #coverpoint_3#	100.00%	100	-	Covered
covered/total bins:	2	2	-	
missing/total bins:	0	2	-	
% Hit:	100.00%	100	-	
bin low	1350	1	-	Covered

bin high 153 1 - Covered

3.5 Assertion coverage

ASSERTION RESULTS:

Name File(Line) Failure Pass

Count Count

/top/DUT/reset_a SPI_wrapper.sv(22) 0 1

/top/DUT/SLAVE_instance/assert__2

SPI_slave.sv(154) 0 1

/top/DUT/SLAVE_instance/assert__1

SPI_slave.sv(149) 0 1

/top/DUT/SLAVE_instance/assert__0

SPI_slave.sv(144) 0 1

/top/DUT/SLAVE_instance/assert_rst_a

SPI_slave.sv(138) 0 1

/top/DUT/SLAVE_instance/write_add_a

SPI_slave.sv(164) 0 1

/top/DUT/SLAVE_instance/write_data_a

SPI_slave.sv(173) 0 1

/top/DUT/SLAVE_instance/read_add_a

SPI_slave.sv(182) 0 1

/top/DUT/SLAVE_instance/read_data_a

SPI_slave.sv(190) 0 1

/top/DUT/inst_sva/MISO_stable_a

SPI_wrapper_sva.sv(11) 0 1

/SPI_wrapper_rd_only_seq_pkg/SPI_wrapper_rd_only_seq/body/#ublk#130736839#15/immed__22

SPI_wrapper_rd_only_seq.sv(22)

0 1

/SPI_wrapper_wr_only_seq_pkg/SPI_wrapper_wr_only_seq/body/#ublk#130738215#15/immed__22

SPI_wrapper_wr_only_seq.sv(22)

0 1

/SPI_wrapper_wr_rd_seq_pkg/SPI_wrapper_wr_rd_seq/body/#ublk#127550407#15/immed__22

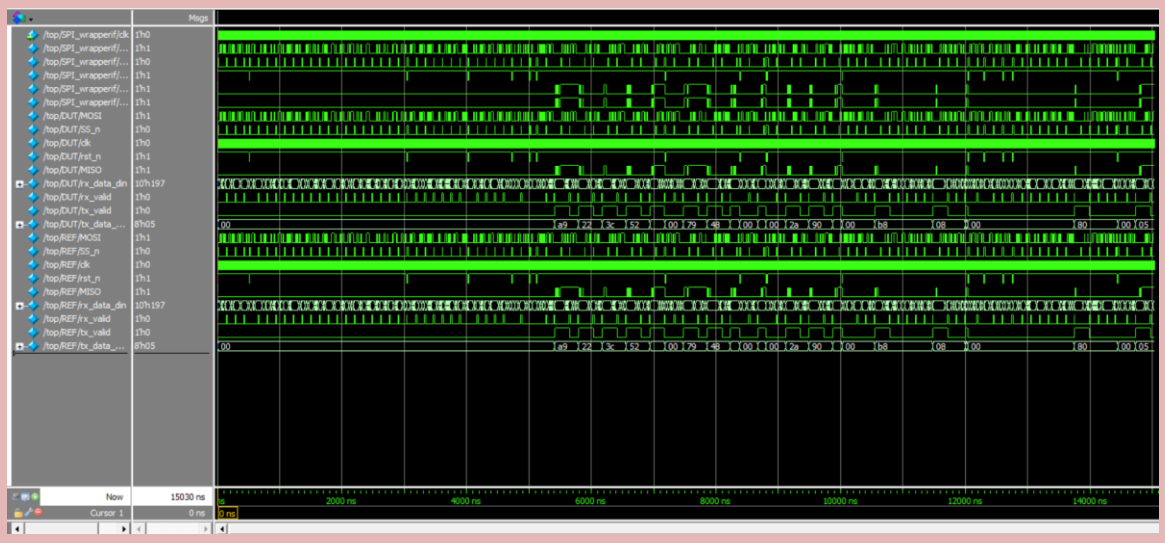
SPI_wrapper_wr_rd_seq.sv(22)

0 1

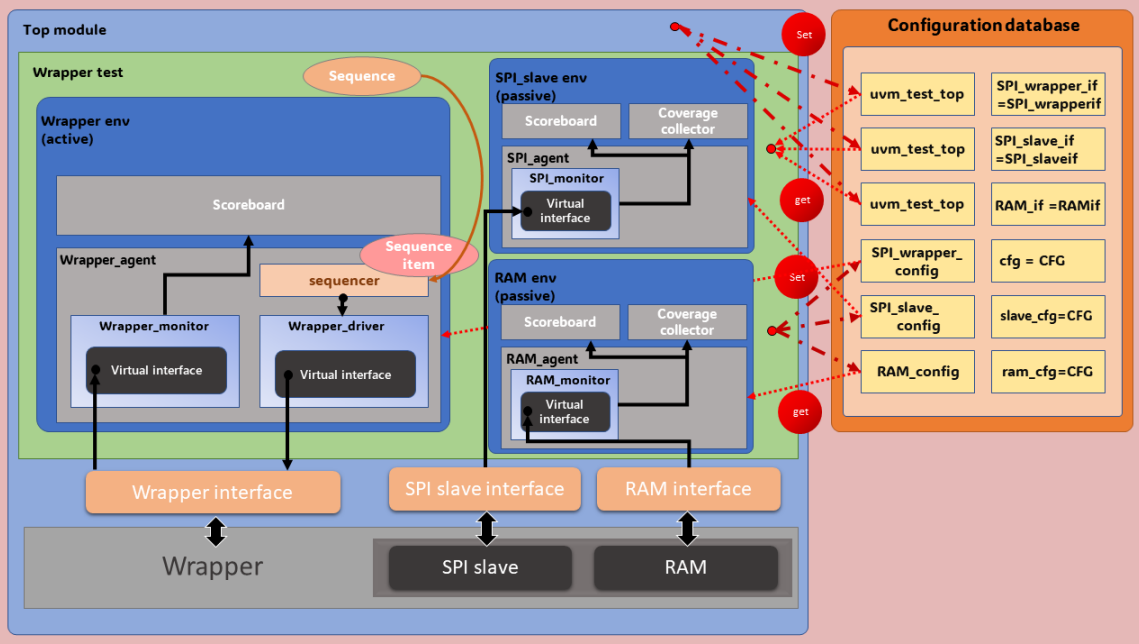
4. Bug report

The Bugs Were Fixed in the slave and ram RTL's

5. Waveform



UVM illustration:



This section details the UVM testbench architecture designed to verify the top-level `Wrapper` DUT. The architecture, shown in the figure above, emphasizes modularity and reuse by integrating a primary active environment with two passive environments from previous project parts.

1. Hierarchical Structure

The testbench is organized into distinct hierarchical layers to ensure separation of concerns and scalability:

- **`Top Module`**: This is the static, outermost container. It instantiates the `Wrapper` DUT, all the necessary physical interfaces (`Wrapper interface`, `SPI slave interface`, `RAM interface`), and the top-level UVM test class (`Wrapper test`).
- **`Wrapper test`**: This is the top-level UVM component. Its primary responsibilities are to instantiate the main verification environment (`Wrapper env`), configure the entire testbench using the `uvm_config_db`, and select and start the test `Sequence`.
- **Environments**: The architecture is composed of three key environments:
    1. **`Wrapper env (active)`**: The main environment responsible for actively driving stimulus to the DUT and performing end-to-end checking.
    2. **`SPI_slave env (passive)`**: A reused environment from Part 1, configured to passively monitor the internal interface between the `Wrapper` and its `SPI slave` instance.
    3. **`RAM env (passive)`**: A reused environment from Part 2, configured to passively monitor the internal interface between the `Wrapper` and its `RAM` instance.

2. Stimulus Generation and Driving Flow

Stimulus is generated and driven to the DUT through the active `Wrapper env`:

1. **`Sequence`**: This component contains the logic for a specific test scenario (e.g., a series of writes followed by reads). It generates high-level transaction objects called `Sequence Items`.
2. **`sequencer`**: The sequence sends these items to the `sequencer` located within the `Wrapper_agent`. The sequencer arbitrates and forwards the transactions to the driver.
3. **`Wrapper_driver`**: The driver pulls sequence items from the sequencer and translates these abstract transactions into pin-level signal activity on the physical `Wrapper interface`, thus driving the DUT.

3. Monitoring and Verification Flow

Verification is achieved through continuous monitoring and checking at multiple points:

- **Monitors**: Each agent contains a monitor responsible for observing signal activity on its respective interface and converting it back into transaction objects.
    - **`Wrapper_monitor`**: Captures transactions on the external DUT interface, observing the stimulus being driven and the DUT's primary response.
    - **`SPI_monitor`** & **`RAM_monitor`**: These reside in the passive agents. They non-intrusively capture the transactions on the internal DUT interfaces, providing visibility into how the `Wrapper` communicates with its sub-modules.
- **Scoreboards**: The architecture employs a hierarchical checking strategy. All transactions captured by the monitors are sent to their respective scoreboards for analysis.
    - The main **`Scoreboard`** in the `Wrapper env` performs end-to-end data integrity checks, comparing the transactions sent by the active driver with the results observed by all three monitors.
    - The scoreboards within the passive environments can be used for protocol-specific checks or local data verification before results are propagated to the top-level scoreboard.
- **Coverage Collectors**: Transactions from the monitors are also sent to `Coverage collectors` to sample functional coverage points, ensuring that the verification plan's goals have been met.

4. Configuration via `uvm_config_db`

The `Configuration database` is used to configure the testbench's structure and connectivity before the test begins:

- **Setting Configuration**: From the top-level `Wrapper test`, `set` calls are used to broadcast configuration information. This includes passing virtual interface handles to the drivers and monitors that need them, and most importantly, setting the activity level of each agent.
- **Getting Configuration**: During the build phase, components like the agents use `get` calls to retrieve their configuration from the database. This is how the `Wrapper_agent` knows to build its driver and sequencer (active mode), while the `SPI_agent` and `RAM_agent` know to build only their monitors (passive mode).

This architecture effectively verifies the integrated `Wrapper` DUT by actively controlling its inputs and outputs while passively observing its internal operations, achieving a high degree of reuse and creating a robust, modular verification environment.